Computational Studies of Entanglement in Spin Chains Involving Fibonacci and Bernoulli Number Sequences

Dissertation submitted to
Visvesvaraya National Institute of Technology, Nagpur
in partial fulfilment of the requirements for the award of degree of

Master of Science
In
Physics

By

Ankan Bhattacharyya
(MS23PHY021)

under the supervision of

Dr. M. S. Ramkarthik
Assistant Professor
Department of Physics,
VNIT Nagpur

Department of Physics
Visvesvaraya National Institute of Technology, Nagpur
2023-2025.

©Visvesvaraya National Institute of Technology (VNIT), 2025

Computational Studies of Entanglement in Spin Chains Involving Fibonacci and Bernoulli Number Sequences

Dissertation submitted to
Visvesvaraya National Institute of Technology, Nagpur
in partial fulfilment of the requirements for the award of degree of

Master of Science
In
Physics

By

Ankan Bhattacharyya
(MS23PHY021)

under the supervision of

Dr. M. S. Ramkarthik
Assistant Professor
Department of Physics,
VNIT Nagpur

Department of Physics
Visvesvaraya National Institute of Technology, Nagpur

2023-2025.

©Visvesvaraya National Institute of Technology (VNIT), 2025

To my parents

Mrs. Antara Bhattacharyya,
Mr. Arindam Bhattacharyya,

&

To my friend

Kaushal Joshi

for their continual support
and endless motivation.

VISVESVARAYA NATIONAL INSTITUTE OF TECHNOLOGY, NAGPUR
DEPARTMENT OF PHYSICS

Dissertation approval sheet

Dissertation entitled **"Computational Studies of Entanglement in Spin Chains Involving Fibonacci and Bernoulli Number Sequences"** by **Ankan Bhattacharyya** is approved for the degree of Master of Science in Physics.

_____ *(Name:)* **Supervisor**	_____ *(Name:)* **External Examiner**

_____
(Name:)
Chairman

Department of Physics

Visvesvaraya National Institute of Technology, Nagpur

Declaration

I, **Ankan Bhattacharyya**, hereby declare that this dissertation titled **"Computational Studies of Entanglement in Spin Chains Involving Fibonacci and Bernoulli Number Sequences"** is carried out by me in the Department of Physics of Visvesvaraya National Institute of Technology, Nagpur. The work is original and has not been submitted earlier, whole or in part, for the award of any degree/diploma at this or any other Institution / University.

Name: Ankan Bhattacharyya

Date: _____

Certificate

This is to certify that the dissertation titled **"Computational Studies of Entanglement in Spin Chains Involving Fibonacci and Bernoulli Number Sequences"** is submitted by **Ankan Bhattacharyya** in partial fulfillment of the requirements for the award of the degree of Master of Science in VNIT Nagpur. The work is comprehensive, complete and fit for final evaluation.

Dr. M. S. Ramkarthik
(Thesis Supervisor)
Asst. Prof., Dept. of Physics,
VNIT, Nagpur

Head, Department of Physics
VNIT, Nagpur
Date: _____

VISVESVARAYA NATIONAL INSTITUTE OF TECHNOLOGY, NAGPUR

Non-Plagiarism Certificate

Certified that M.Sc. Dissertation titled **"Computational Studies of Entanglement in Spin Chains Involving Fibonacci and Bernoulli Number Sequences"** submitted by **Ankan Bhattacharyya**, Enrollment No. MS23PHY021, ID No. 30704, is original in nature and not copied from any literature material. Due credit has been given to the original researcher wherever referred.

Signature:

Name of the student: Ankan Bhattacharyya

Enrollment No.: MS23PHY021

Signature:

Dr. M. S. Ramkarthik (Asst. Prof., Dept. of Physics, VNIT)

MSc Thesis

ORIGINALITY REPORT

17%	5%	16%	0%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

ACKNOWLEDGMENTS

Firstly, I would like to express my sincere gratitude to my guide, ***Dr. M. S. Ramkarthik****, for allowing me to do this project work under his supervision. His dedication towards his work and his proficiency in mentoring students have been a great source of motivation for me. It gives me inspiration on how to become an ideal teacher. The Saturday discussions that were used to organized by him every week during our group meetings helped me a lot in understanding various fascinating topics related to the physical world.*

I want to acknowledge my parents, ***Mrs. Antara Bhattacharyya*** *and* ***Mr. Arindam Bhattacharyya****, for their unwavering support and for giving priority to my decision. Next, I would like to acknowledge my friend* ***Kaushal Joshi*** *for motivating me a lot whenever I felt frustrated. He used to help me in resolving the technical glitches while working on this project. I appreciate his support and the insightful discussions that we used to have during these two years of M.Sc .*

I want to acknowledge my group members, Gargee Arun Tammadwar, Divyani Vilas Bhagat, Durgesh Thail Vasant, Om Sanjay Shah, and Atul. The discussions that we all used to have together are priceless.

I also want to express my heartfelt gratitude to all of my friends. These past two years were nothing but a sort of beautiful journey, filled with laughter, support, and unforgettable moments. Each one of them has played a vital role in shaping my experience. Thanks to all of them for being an incredible part of my life.

ABSTRACT

Quantum entanglement is a strange and exclusive phenomenon in the quantum world, where two or more systems become strongly correlated such that their quantum state cannot be described independently, even when they are separated by a large distance. Nowadays, it has become a very attractive topic for research because of its immense application in quantum computing and information tasks. In this project work, we have tried to study the entanglement in different spin chain systems that involve the nearest neighbour interaction, followed by some particular number sequences, which are the Fibonacci and Bernoulli sequences of numbers. We are proposing this type of spin chain involving inverse Fibonacci and Bernoulli numbers for the first time. These systems are possible candidates to explain quasicrystals, etc.

We have started from the introduction to the spin chains and the methods by which we can easily construct the Hamiltonians for such types of spin chains numerically. We have also studied how to get the eigenvalue spectrum corresponding to such types of spin chains numerically. The introduction to these number series, viz. Fibonacci and Bernoulli are also provided, which will help the reader to revisit these number series and find some important properties of them, which will be later used in this thesis.

The entanglement studies corresponding to the qubit pairs are done for the above spin chains. The relation between the two number series, which we have expressed in this thesis, indicates that there must be some relation in the entanglement also. With this aim in mind, we have found and analyzed our results to establish a relation between the concurrences corresponding to the qubit pairs for the two number-theoretic spin chains. Multiqubit entanglement properties of the aforesaid spin chains are also studied using entropic measures to see any possible entanglement build up.

List of Figures

1.1 One-dimensional Ising model with open chain condition 2

1.2 One-dimensional Ising Model with Periodic boundary condition . . . 3

1.3 Flowchart illustrating the variation of energy eigenvalues with interaction energies and magnetic field. 18

1.4 Variation of energy eigenvalue with magnetic field in the one-dimensional Ising model for case(1). 19

1.5 Variation of energy eigenvalue with interaction energy in the one-dimensional Ising model for case(1). 19

1.6 Variation of energy eigenvalues with magnetic field in the one-dimensional Ising model for a 3-qubit system considering case(2). 20

1.7 Variation of energy eigenvalues with interaction energy in the one-dimensional Ising model for 3 qubit system considering case(2). . . . 20

1.8 Homonuclear molecule consisting of two one-electron atoms 21

1.9 Spin Configuration for (a) Symmetric and (b) Antisymmetric arrangement 22

2.1 Golden ratio of Fibonacci series 36

3.1 Penrose tiling [16] 41

3.2 Open chain system 43

3.3 Periodic chain system 43

3.4 Variation of energy eigenvalue with Magnetic field for an open chain system consisting of three qubits when the interaction energies are taken along one direction. 49

3.5 Variation of energy eigenvalue with magnetic field for a periodic chain system consisting of three qubits when the interaction energies are taken along one direction 55

4.1 Interaction of the two boys for some time. 58

4.2 Pictorial analogy of entanglement between the two boys 58

4.3 Quantum Teleportation 59

4.4 Division of the total system into two blocks. 61

4.5 Monogamy of Entanglement 62

4.6 Variation of Concurrences for the 3-qubit open chain Ising-like System 74

4.7 Concurrence Study for 3-qubit periodic chain Ising-like System 93

4.8 Division of the total system into two blocks. 102

4.9 Block entropy study for 3 qubits open chain transverse Ising-like system with inverse Fibonacci type of nearest neighbour interaction. . . 113

4.10 Block entropy Study for 3 qubit open chain system 131

4.11 Study of the monogamy of entanglement for a three-qubit open-chain Ising-like system with inverse Fibonacci interaction. 139

4.12 Study of the monogamy of entanglement for three-qubit periodic chain Ising-like systems with inverse Fibonacci interaction. 140

5.1 Energy eigenvalue spectrum with variation of magnetic field for Heisenberg-Fibonacci spin chain system consisting of 3 qubits 153

5.2 Energy eigenvalue spectrum with variation of magnetic field for Heisenberg-like periodic spin chain system consisting of 3 qubits 162

5.3 Variation of Concurrences with magnetic field for three-qubit open chain Heisenberg-like system. 176

5.4 Variation of Concurrences with magnetic field for four-qubit open chain Heisenberg-like system. 176

5.5 Variation of ground state energy with magnetic field for a 4-qubit open chain system corresponding to the Heisenberg-Fibonacci model. . . . 177

5.6 Concurrences obtained from correlation function and that obtained earlier for 4 qubit open chain Heisenberg-like spin chain system. . . . 196

5.7 Variation of Concurrences with magnetic field for three-qubit periodic chain Heisenberg-Fibonacci system. 215

5.8 Variation of Concurrences with magnetic field for four-qubit periodic chain Heisenberg-like system. 216

5.9 Variation of Concurrences with the Magnetic Field corresponding to 8-qubit open chain Heisenberg-like system. 221

5.10 Variation of von Neumann entropy with magnetic field for three-qubit open chain Heisenberg-Fibonacci system. 233

5.11 Variation of von Neumann entropy with magnetic field for four-qubit open chain Heisenberg-Fibonacci system. 234

5.12 Variation of von Neumann entropy with magnetic field for three-qubit periodic chain Heisenberg-Fibonacci system. 251

5.13 Variation of von Neumann entropy with magnetic field for four-qubit periodic chain Heisenberg-Fibonacci system. 252

5.14 Variation of von Neumann Entropy with the Magnetic Field corresponding to 8-qubit periodic chain Heisenberg-like system. 257

5.15 Monogamy of Entanglement(MOE) for three-qubit open chain Heisenberg-Fibonacci system. 258

5.16 Monogamy of Entanglement(MOE) for three-qubit periodic chain Heisenberg-Fibobsnni system. 258

6.1 Plotting of the first 14 Bernoulli numbers with the number of terms. . 276

7.1 Open chain system 286

7.2 Periodic chain system 286

7.3 Energy eigenvalue spectrum with variation of magnetic field for 3 qubits open chain system 294

7.4 Energy eigenvalue spectrum with variation of magnetic field for 3 qubits periodic chain system 301

7.5 Variation of Concurrences with magnetic field for three-qubit open chain system. 312

7.6 Variation of Concurrences with magnetic field for three-qubit open chain system (For both Inverse Fibonacci and Bernoulli type interaction).313

7.7 Variation of Concurrences with magnetic field for three-qubit periodic chain system. 324

7.8 Variation of Concurrences with magnetic field for three-qubit periodic chain system (For both Inverse Fibonacci and Bernoulli type interaction).324

7.9 Variation of von Neumann entropy with the magnetic field for a three-qubit open chain system. 334

7.10 Variation of von Neumann entropy with magnetic field for a three-qubit open chain system (For both Inverse Fibonacci and Bernoulli type interaction). 335

7.11 Variation of von Neumann entropy with the magnetic field for three-qubit periodic chain system. 345

7.12 Variation of von Neumann entropy with magnetic field for three-qubit periodic chain system (For both Inverse Fibonacci and Bernoulli type interaction). 346

7.13 Monogamy of Entanglement(MOE) for three-qubit open chain system. 347

7.14 Monogamy of Entanglement(MOE) for three-qubit periodic chain system. 347

8.1 Interpolation of ground state energies obtained from Bernoulli and inverse Fibonacci interactions. 367

8.2 Interpolation of concurrences corresponding to inhomogeneous transverse Ising-like 3 qubit open chain system by considering two sets of interaction schemes. 373

8.3 Floral Function in Polar Coordinates. 374

8.4 Floral Function in Cartesian Coordinates. 375

8.5 Effect of fitting parameter K in floral function. 376

8.6 Effect of fitting parameter l in floral function. 377

8.7 Effect of fitting parameter ϕ in floral function. 378

8.8 Fitting the C_{12} data with the help of floral function. 379

8.9 Zoomed in view of fitted portion of the C_{12} data. 379

8.10 Fitting the C_{23} data with the help of floral function. 380

8.11 Zoomed in view of fitted portion of the C_{23} data. 381

8.12 Fitting the C_{13} data with the help of floral function. 382

8.13 Zoomed in view of fitted portion of the C_{13} data. 382

List of Tables

1.1 Two different cases for the one-dimensional Ising model with different physical properties. 9

2.1 Number of rabbit pairs showing the Fibonacci sequence. 28

2.2 Convergence of Ratio Between Successive Terms 32

8.1 Ground state energies for Bernoulli and inverse Fibonacci interactions 365

8.2 Concurrences between pairs of qubits for Bernoulli and Inverse Fibonacci chains. 373

Contents

1 **Introduction to Spin Chain Systems via One-Dimensional Ising Model and One-Dimensional Heisenberg Model** **1**

1.1 Introduction to One-Dimensional Ising Model 1

1.1.1 Solution of the One-Dimensional Ising Model 3

1.1.1.1 Using the Tensor product approach 3

1.1.1.2 Elementwise study of Hamiltonian matrix through Bit test Method 8

1.2 Introduction to One-Dimensional Heisenberg Model 21

1.3 Chapter Summary 25

2 **Fibonacci sequence and Its Importance** **27**

2.1 Introduction 27

2.2 Fibonacci Sequence and Golden Ratio 28

2.2.1 Golden Ratio obtained for 100 numbers of the Fibonacci Series 30

2.3 Importance of Fibonacci Sequence 36

2.4 Reciprocal Fibonacci series 37

2.5 Chapter Summary 39

3 Transverse Ising-like model with the Inverse Fibonacci type of Interactions 40

3.1 Introduction 40

3.1.1 What is Quasicrystal? 41

3.1.2 Mathematical construction of Quasicrystal-like system 42

3.2 Transverse Ising-like model with the Inverse Fibonacci type of Interactions 42

3.2.1 Study of the Energy Eigenvalue Spectrum with the variation of Magnetic field 44

3.2.1.1 Case-(1) Open Chain System 44

3.2.1.2 Case-(2) Periodic Chain System 50

3.3 Chapter Summary 56

4 Entanglement Study in Ising-like Spin Chain systems 57

4.1 Introduction to Quantum Entanglement 57

4.1.0.1 Importance of Quantum Entanglement 59

4.2 Entanglement Measure 59

4.2.1 Concurrence 60

4.2.2 von Neumann Entropy and Block Entropy 60

4.2.3 Monogamy Of Entanglement(MOE) 62

4.3 Entanglement Study in Transverse Ising-like model with the Inverse Fibonacci type of Interactions 62

4.3.1 Entanglement Study for systems consisting of N qubits 63

4.3.1.1 **Case-(1) For Open Chain System** 63

4.3.1.2 Concurrence study for 12 qubit open chain Ising-like system 75

4.3.1.3 **Case-(2) For Periodic Chain System** 82

4.3.1.4 Concurrence study for 12 qubit periodic chain Ising-like system 95

4.3.2 Study of Block Entropy for the System 102

4.3.2.1 **Case-(1) For Open Chain System** 103

4.3.2.2 Block entropy study for 12 qubit open chain Ising-like system 114

4.3.2.3 **Case-(2) For Periodic Chain System** 121

4.3.2.4 Block entropy study for 12 qubit periodic chain Ising-like system 132

4.3.3 Monogamy Of Entanglement(MOE) 139

4.4 Chapter Summary 140

5 Entanglement Study in Heisenberg-type Spin Chain systems 142

5.1 Introduction 142

5.2 Numerical Solution of Heisenberg-like Spin Chain System Consisting of N Number of qubits 142

5.2.1 Case-(1):Open Chain System 144

5.2.2 Case-(2):Periodic Chain System 154

5.3 Entanglement Study in Heisenberg-Fibonacci Spin chain system . . . 162

5.3.1 Studies on Concurrence 162

5.3.1.1 Case(1): Open chain System 163

5.3.1.2 Reason for the discontinuity in concurrences 177

5.3.1.3 Numerical Verification 181

5.3.1.4 Concurrences for 8 qubit open Heisenberg-Fibonacci spin chain system 198

5.3.1.5 Case(2): Periodic chain System 202

5.3.1.6 Concurrences for 8 qubit periodic Heisenberg-Fibonacci spin chain system 217

5.3.2 Studies on von-Neumann Entropy 221

5.3.2.1 Case(1): Open Chain System 221

5.3.2.2 Block entropies for 8 qubit open Heisenberg-Fibonacci spin chain system 235

5.3.2.3 Case(2): Periodic Chain System 239

5.3.2.4 Block entropies for 8 qubit periodic Heisenberg-Fibonacci spin chain system 253

5.3.3 Monogamy of Entanglement(MOE) 257

5.4 Chapter Summary 258

6 Introduction to Bernoulli Numbers and Relation between Fibonacci and Bernoulli polynomials 260

6.1 Introduction 260

6.2 What are Bernoulli numbers? 261

6.3 Determination of Bernoulli numbers from the generating function . . 264

6.4 Recurrence relation for Bernoulli number 272

6.5 Numerical generation of Bernoulli Numbers 273

6.5.0.1 List of first 14 Bernoulli numbers 275

6.6 Relations and Identities of Bernoulli Polynomials 276

6.6.1 Generating Function Of Bernoulli Number and Bernoulli Polynomial 276

6.6.2 Relation Between Bernoulli Polynomial and Bernoulli Number 277

6.6.3 An interesting Relation 278

6.6.4 Recursion Relation of the Bernoulli polynomial 278

6.6.5 Inversion Formula in the Continuous case 278

6.6.6 Translation formula for Bernoulli Polynomials 279

6.6.7 Inversion Formula in the discrete case 280

6.7 Fibonacci polynomial 281

6.8 Relation between Fibonacci and Bernoulli polynomials 282

6.9 Chapter Summary 283

7 One dimensional Ising-like Model with Bernoulli type of interaction and Entanglement Studies 285

7.1 Introduction 285

7.2 Study of Eigenvalue Spectrum for two different Cases 286

7.2.1 Case(1) Open Chain System 287

7.2.2 Case(2) Periodic Chain System 294

7.3 Entanglement Study 301

7.3.1 Study of Concurrence 301

7.3.1.1 Case(1) Open Chain System 302

7.3.1.2 Case(2) Periodic Chain System 313

7.3.2 Study of von Neumann Entropy 325

7.3.2.1 Case(1) Open Chain System 325

7.3.2.2 Case(2) Periodic Chain System 335

7.4 Monogamy of Entanglement (MOE) 346

7.5 Chapter Summary 348

8 Analysis of The Numerical Results 349

8.1 Introduction 349

8.2 Relation between Fibonacci and Bernoulli numbers 349

8.2.0.1 Case(1) For some low value of n (e.g. $n = 8$) 354

8.2.0.2 Case(2) For some medium value of n (e.g. $n = 30$) . 355

8.2.0.3 Case(3) For some large value of n (e.g. $n = 70$) . . . 356

8.3 Numerical study to determine the relation between the results obtained from the two different transverse Ising-like Models involving Bernoulli and inverse Fibonacci type of interactions 360

8.3.1 Relation between the Ground state Energies of the two systems 360

8.3.2 Relation between the Concurrences 368

8.3.2.1 Fitting of C_{12} data points with the help of floral function. 378

8.3.2.2 Fitting of C_{23} data points with the help of floral function. 380

8.3.2.3 Fitting of C_{13} data points with the help of floral function. 381

8.4 Chapter Summary 383

Chapter 1

Introduction to Spin Chain Systems via One-Dimensional Ising Model and One-Dimensional Heisenberg Model

---

1.1 Introduction to One-Dimensional Ising Model

At first, we are going to discuss a very simple but tremendously important spin chain model, which is the Ising Model, and try to solve the model for the one-dimensional case. It is well known to us that the Ising model is a mathematical model that can be used to describe the phase transition in a system [1]. It was mainly modeled by Ernst Ising to discuss the "spontaneous magnetization" of ferromagnetic materials, which was the subject of his doctoral dissertation. This model can be given in the following way. Suppose we are considering a linear one-dimensional lattice consisting of N sites, each of which contains a spin that may be either up or down, and the Hamiltonian $\mathcal{H}$ is given by as follows,

$$\mathcal{H} = -J\sum_i \sigma_i^z \sigma_{i+1}^z - B\sum_i \sigma_i^z \,. \tag{1.1}$$

Now we may think that the Ising model is biased to the z-direction only, but it can be true for the x-direction as well as the y-direction, which can be given as follows.

$$\mathcal{H} = -J\sum_i \sigma_i^x \sigma_{i+1}^x - B\sum_i \sigma_i^x, \tag{1.2}$$

$$\mathcal{H} = -J\sum_i \sigma_i^y \sigma_{i+1}^y - B\sum_i \sigma_i^y. \tag{1.3}$$

Here, σ_i's given as

$$\sigma^x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \sigma^y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \sigma^z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \tag{1.4}$$

which are the x,y,z-components of the Pauli spin matrix, and J represents the spin-spin interaction, which is varied in the same way between the successive neighbors, B represents the external magnetic field.

The limit on the first sum is not mentioned in Eq.[1.3] to allow us to describe two kinds of boundary conditions, which are as follows:
(1) ***Open chain condition*** - where the summation corresponding to the first term extends from $i = 1$ to $i = N - 1$ as shown below.

Figure 1.1: One-dimensional Ising model with open chain condition

The Hamiltonian from Eq.[1.3] can now be expressed for this case as,

$$\mathcal{H} = -J\sum_{i=1}^{N-1} \sigma_i^a \sigma_{i+1}^a - B\sum_{i=1}^{N} \sigma_i^a. \tag{1.5}$$

where a may be $x, y,\ orz$, representing the direction of spin angular momentum in three-dimensional cartesian systems. In this case, the first and the last sites are not connected.

(2) ***Periodic boundary condition*** - In this case, $\sigma_{N+1} \equiv \sigma_1$ is used to consider, and the first sum is allowed to go from $i = 1$ to N. It corresponds to a chain interaction around a closed loop or a wrap-up. The Hamiltonian can now be expressed

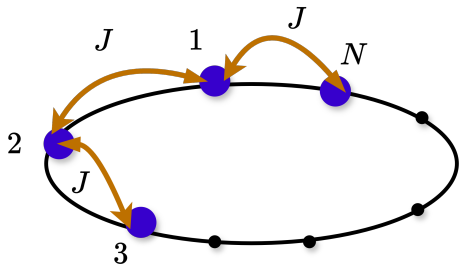


Figure 1.2: One-dimensional Ising Model with Periodic boundary condition

as,

$$\mathcal{H} = -J \sum_{i=1}^{N} \sigma_i^a \sigma_{i+1}^a - B \sum_{i=1}^{N} \sigma_i^a \tag{1.6}$$

where a may be $x, y,$ *or* z, representing the direction of spin angular momentum in a three-dimensional cartesian system. In both cases, the second sum can go from $i = 1$ to $i = N$ to include all the spins.

1.1.1 Solution of the One-Dimensional Ising Model

The one-dimensional Ising model is exactly solvable, and we will discuss some interesting results coming from it by considering different cases.

1.1.1.1 Using the Tensor product approach

In this section, we demonstrate how to calculate the energy eigenvalues of the Ising model for a small number of spins (referred to as qubits, where qubit '0' represents an 'up' spin and qubit '1' represents a 'down' spin, this is done to take the leverage of the binary to decimal representation). Let us consider a system comprising three qubits. The Hamiltonian corresponding to the system is given by:

$$\mathcal{H} = -J \sum_{i=1}^{3} \sigma_i^z \sigma_{i+1}^z - B \sum_{i=1}^{3} \sigma_i^z, \tag{1.7}$$

where periodic boundary conditions are applied such that $\sigma_4 = \sigma_1$, and we have considered that the spin-spin interaction as well as the direction of applied magnetic field B is in the same direction as in the Z direction.

To solve it, the Hamiltonian operator can be expressed as:

$$\mathcal{H} = -J[\sigma_1^z \otimes \sigma_2^z \otimes \mathbf{I} + \mathbf{I} \otimes \sigma_2^z \otimes \sigma_3^z + \sigma_3^z \otimes \mathbf{I} \otimes \sigma_1^z] - B[\sigma_1^z \otimes \mathbf{I} \otimes \mathbf{I} + \mathbf{I} \otimes \sigma_2^z \otimes \mathbf{I} + \mathbf{I} \otimes \mathbf{I} \otimes \sigma_3^z], \quad (1.8)$$

where σ_i's are the Pauli matrices and $\mathbf{I}$ is the 2×2 identity matrix. The eigenvalues of this system can be computed using the following Fortran code:

```
program Ising_model_for_three_qubit_system
    implicit none
   integer :: sig1(2,2), sig2(2,2), sig3(2,2), P1(4,4), &
           I(2,2), P2(4,4), P3(4,4), P4(4,4), P5(4,4), P6(4,4)

    real*8 :: S1(8,8), S2(8,8), S3(8,8), S4(8,8), S5(8,8), S6(8,8),
   H(8,8)
    integer :: a, b
    real*8 :: Y, X

    ! Parameters for LAPACK DSYEV
    integer :: lda, info, lwork
    real*8 :: w(8), work(24)
    character*1 :: JOBZ, UPLO
    lda = 8
    lwork = 24

    ! Parameters for Hamiltonian
    print*, 'Enter the value of J'
    read*, Y
    print*, 'Enter the value of a magnetic field'
    read*, X

    ! Initialization of sig1
    sig1(1,1) = 1
    sig1(1,2) = 0
    sig1(2,1) = 0
    sig1(2,2) = -1
```

```
28      ! Initialization of sig2
29      sig2(1,1) = 1
30      sig2(1,2) = 0
31      sig2(2,1) = 0
32      sig2(2,2) = -1
33      ! Initialization of sig3
34      sig3(1,1) = 1
35      sig3(1,2) = 0
36      sig3(2,1) = 0
37      sig3(2,2) = -1
38      ! Initialization of Identity matrix I
39      I(1,1) = 1
40      I(1,2) = 0
41      I(2,1) = 0
42      I(2,2) = 1
43
44      ! Find P1(4,4) using tensor product of sig1 and sig2
45      call Tensorprod(sig1, sig2, P1, 2, 2, 2, 2)
46
47      ! Find S1 using tensor product of P1(4,4) and I(2,2)
48      call Tensorprod_real(P1, I, S1, 4, 4, 2, 2)
49
50      ! Find P2(4,4) using tensor product of I and sig2
51      call Tensorprod(I, sig2, P2, 2, 2, 2, 2)
52
53      ! Find S2 using tensor product of P2(4,4) and sig3
54      call Tensorprod_real(P2, sig3, S2, 4, 4, 2, 2)
55
56      ! Find P3(4,4) using tensor product of sig3 and I
57      call Tensorprod(sig3, I, P3, 2, 2, 2, 2)
58
59      ! Find S3 using tensor product of P3(4,4) and sig1
60      call Tensorprod_real(P3, sig1, S3, 4, 4, 2, 2)
61
62      ! Find P4(4,4) using tensor product of sig1 and I
63      call Tensorprod(sig1, I, P4, 2, 2, 2, 2)
64
65      ! Find S4 using tensor product of P4(4,4) and I
```

```
66      call Tensorprod_real(P4, I, S4, 4, 4, 2, 2)
67
68      ! Find P5(4,4) using tensor product of I and sig2
69      call Tensorprod(I, sig2, P5, 2, 2, 2, 2)
70
71      ! Find S5 using tensor product of P5(4,4) and I
72      call Tensorprod_real(P5, I, S5, 4, 4, 2, 2)
73
74      ! Find P6(4,4) using the tensor product of I and I
75      call Tensorprod(I, I, P6, 2, 2, 2, 2)
76
77      ! Find S6 using tensor product of P6(4,4) and sig3
78      call Tensorprod_real(P6, sig3, S6, 4, 4, 2, 2)
79
80      ! Final Hamiltonian
81      H = Y * (S1 + S2 + S3) + X * (S4 + S5 + S6)
82
83      ! Eigenvalue and eigenvector determination using DSYEV
84      JOBZ = 'V'
85      UPLO = 'U'
86      call DSYEV(JOBZ, UPLO, lda, H, lda, w, work, lwork, info)
87
88      if (info == 0) then
89          print*, 'Eigenvalues:'
90          do a = 1, lda
91              print*, 'W(', a, ' )= ', w(a)
92          end do
93      else
94          print*, 'Error in DSYEV, INFO=', info
95      end if
96
97  contains
98
99      subroutine Tensorprod(A, B, C, m, n, p, q)
100         implicit none
101         integer, intent(in) :: m, n, p, q
102         integer, intent(in) :: A(m, n), B(p, q)
103         integer, intent(out) :: C(m * p, n * q)
```

```
104         integer :: i, j, k, l
105
106         do i = 1, m
107             do j = 1, n
108                 do k = 1, p
109                     do l = 1, q
110                         C((i - 1) * p + k, (j - 1) * q + l) = A(i, j
    ) * B(k, l)
111                     end do
112                 end do
113             end do
114         end do
115     end subroutine Tensorprod
116
117     subroutine Tensorprod_real(A, B, C, m, n, p, q)
118         implicit none
119         integer, intent(in) :: m, n, p, q
120         integer, intent(in) :: A(m, n), B(p, q)
121         real*8, intent(out) :: C(m * p, n * q)
122         integer :: i, j, k, l
123
124         do i = 1, m
125         do j = 1, n
126             do k = 1, p
127                 do l = 1, q
128                    C((i - 1) * p + k, (j - 1) * q + l) = real(A(i, j
    )) * real(B(k, l))
129                 end do
130             end do
131         end do
132         end do
133     end subroutine Tensorprod_real
134
135 end program Ising_model_for_three_qubit_system
```

The above code constructs the Hamiltonian matrix(for the three-qubit system) using tensor products and computes its eigenvalues using LAPACK's DSYEV subroutine.

Each eigenvalue corresponds to an energy level of the system. We can compute the Hamiltonian for a higher number of qubits using this method. Still, we have already seen for three qubits, the number of tensor products is three and the order of the matrix corresponding to the Hamiltonian is 8 x 8, i.e. $2^3 \times 2^3$. For a general n number of qubits, the order of the matrix will be $2^n \times 2^n$, which will require much more computational time as well as memory in the computer. We can follow a much smarter approach to make our job easier for a larger number of qubits as follows.

1.1.1.2 Elementwise study of Hamiltonian matrix through Bit test Method

In this method, we are going to construct the Hamiltonian matrix by finding each of its elements. For that case, we have to know the general operation of Pauli's spin matrices on the qubits like,

$$\begin{aligned} \sigma^z|0\rangle = |0\rangle \qquad \sigma^z|1\rangle = -|1\rangle, \\ \sigma^x|0\rangle = |1\rangle \qquad \sigma^x|1\rangle = |0\rangle. \end{aligned} \tag{1.9}$$

So if you consider a general three-qubit state as $|a_1a_2a_3\rangle$, you can find the general expression of the above operations as,

$$\begin{aligned} \sigma_1^z|a_1a_2a_3\rangle = (1-2a_1)|a_1a_2a_3\rangle, \\ \sigma_1^x|a_1a_2a_3\rangle = |(1-a_1)\ a_2\ a_3\rangle. \end{aligned} \tag{1.10}$$

We can construct the Hamiltonian by utilizing the above properties. We often consider the two different cases for the Hamiltonian -

Case (1): Isotropic periodic chain Ising Model in a non-zero magnetic field	**Case (2): Transverse periodic chain Ising Model in a non-zero magnetic field**
When we consider the spin-spin interaction and the applied magnetic field in the same direction (e.g., along the Z direction)[2]. The Hamiltonian of the system can be given by,	Considering the spin-spin interaction along the Z direction and the applied magnetic field in the X direction [3]. The Hamiltonian of the system can be given by,
$\mathcal{H} = -J\sum_{i=1}^{N}\sigma_i^z\sigma_{i+1}^z - B\sum_{i=1}^{N}\sigma_i^z$	$\mathcal{H} = -J\sum_{i=1}^{N}\sigma_i^z\sigma_{i+1}^z - B\sum_{i=1}^{N}\sigma_i^x$

Table 1.1: Two different cases for the one-dimensional Ising model with different physical properties.

Let us take a three-qubit system for example to show how to form the Hamiltonian. The Hamiltonian as expressed in the 3 qubit basis is,

$$\mathcal{H}= \begin{bmatrix} \langle 000|H|000\rangle & \langle 000|H|001\rangle & \langle 000|H|010\rangle & . & . & . & . & . & \langle 000|H|111\rangle \\ \langle 001|H|000\rangle & \langle 001|H|001\rangle & \langle 001|H|010\rangle & . & . & . & . & . & \langle 001|H|111\rangle \\ \langle 010|H|000\rangle & \langle 010|H|001\rangle & \langle 010|H|010\rangle & . & . & . & . & . & \langle 010|H|111\rangle \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ \langle 111|H|000\rangle & \langle 111|H|001\rangle & \langle 111|H|010\rangle & . & . & . & . & . & \langle 111|H|111\rangle \end{bmatrix}$$

Considering periodic chain conditions,

$$H = -J[\sigma_1^z\sigma_2^z + \sigma_2^z\sigma_3^z + \sigma_3^z\sigma_1^z] - B[\sigma_1^z + \sigma_2^z + \sigma_3^z] \qquad \text{[for the case(1)]}$$

$$H = -J[\sigma_1^z\sigma_2^z + \sigma_2^z\sigma_3^z + \sigma_3^z\sigma_1^z] - B[\sigma_1^x + \sigma_2^x + \sigma_3^x] \qquad \text{[for the case(2)]}$$

The Fortran code below is for calculating the energy eigenvalues by considering the interaction energy J and magnetic field B as 1 unit in case(1) above.

```
program Ising_model_for_n_number_of_qubit
    !Periodic chain condition
    implicit none
    integer :: n, i, num, k, a
    real*8 :: p, B
    character(len=:), allocatable :: state(:)
    integer, allocatable :: decimal(:)
    integer, allocatable :: psi(:)
    integer :: j
    real*8, allocatable :: S1(:,:), S2(:,:), H(:,:), H1(:,:)

    ! Parameters for LAPACK DSYEV
    integer :: lda, info, lwork
    real*8, allocatable :: w(:), work(:)
    character*1 :: JOBZ, UPLO

    ! Input values
    print*, "Enter the number of bits you want:"
    read*, n
    !print*, "Enter the value of B:"
    !read*, B

    ! Calculate the number of states
    num = 2**n
    ! Allocate arrays after determining the value of num
    allocate(character(len=n) :: state(num))
    allocate(integer :: decimal(num))
    allocate(integer :: psi(n))
    allocate(real(8) :: S1(num, num))
    allocate(real(8) :: S2(num, num))
    allocate(real(8) :: H(num, num))
    allocate(real(8) :: H1(num, num))

    ! LAPACK parameters
    lda = num
```

```
    lwork = 3 * num - 1
    allocate(real(8) :: w(lda), work(lwork))

    do i = 0, num - 1
        call integer_binary(i, state(i + 1), n)  ! Adjusting index for 1-based
        !print *, state(i + 1)
    end do

    ! Convert binary states to decimal
    do i = 1, num
        decimal(i) = btod(state(i), n)
    end do

    ! Perform calculations based on the decimal values for S1
    do i = 1, num
        do j = 1, num
            call single_digit(state(i), psi)
            if (decimal(i) /= decimal(j)) then
                S1(i, j) = 0.0
            else
                S1(i, j) = 0.0
                do k = 1, n - 1
                    S1(i, j) = S1(i, j) + (1.0 - 2.0 * psi(k)) * (1.0 - 2.0 * psi(k + 1))
                end do
                S1(i, j) = S1(i, j) + (1.0 - 2.0 * psi(n)) * (1.0 - 2.0 * psi(1))
            end if
        end do
    end do

    ! Perform calculations based on the decimal values for S2
    do i = 1, num
        do j = 1, num
            call single_digit(state(i), psi)
            if (decimal(i) /= decimal(j)) then
                S2(i, j) = 0.0
```

```
            else
                S2(i, j) = 0.0
                do k = 1, n
                    S2(i, j) = S2(i,j) + (1.0 - 2.0 * psi(k))
                end do
            end if
        end do
    end do

    ! Total Hamiltonian
    H =  S1 +  S2

    ! Eigenvalue and eigenvector determination using DSYEV
    JOBZ = 'V'
    UPLO = 'U'
    call DSYEV(JOBZ, UPLO, lda, H, lda, w, work, lwork, info)   if (
   info == 0) then
        !write(722,*) 'Eigenvalues for J=', p
        write(722,*) "Eigenvalue"
        do a = 1, lda
            write(722,*) 'W=(',a,')',w(a)
        end do
    else
        write(722,*) 'Error in DSYEV, INFO=', info
    end if
contains
    function btod(state, n) result(decimal)
        implicit none
        integer, intent(in) :: n
        character(len=n) :: state
        integer :: decimal, i
        decimal = 0
        do i = 1, n
            if (state(i:i) == '1') then
                decimal = decimal + 2**(n-i)
            end if
        end do
    end function btod
```

```
108
109     subroutine integer_binary(num, binary_string, n)
110         implicit none
111         integer :: num
112         integer, intent(in) :: n
113         integer :: i, temp_num
114         character(len=n), intent(out) :: binary_string
115         temp_num = num
116         binary_string = repeat('0', n)
117         do i = 0, n - 1
118             if (mod(temp_num, 2) == 1) then
119                 binary_string(n-i:n-i) = '1'
120             else
121                 binary_string(n-i:n-i) = '0'
122             end if
123             temp_num = temp_num / 2
124         end do
125     end subroutine integer_binary
126
127     subroutine single_digit(A, psi)
128         implicit none
129         character(len=*), intent(in) :: A
130         integer, intent(out) :: psi(:)
131         integer :: i
132         do i = 1, len(A)
133             if (A(i:i) == '0') then
134                 psi(i) = 0
135             else if (A(i:i) == '1') then
136                 psi(i) = 1
137             else
138                 print *, "Invalid input in single_digit subroutine"
139             end if
140         end do
141     end subroutine single_digit
142 end program Ising_model_for_n_number_of_qubit
```

We can also find the energy eigenvalues for **case (2)**, for which we have to change

the S2 matrix of the previous code as follows.

```
program Ising_model_for_n_number_of_qubit
!Periodic chain condition
implicit none
integer :: n, i, num, k, a
real*8 :: p, B
character(len=:), allocatable :: state(:)
integer, allocatable :: decimal(:)
integer, allocatable :: psi(:)
integer :: j
real*8, allocatable :: S1(:,:), S2(:,:), H(:,:), H1(:,:)

! Parameters for LAPACK DSYEV
integer :: lda, info, lwork
real*8, allocatable :: w(:), work(:)
character*1 :: JOBZ, UPLO

! Input values
print*, "Enter the number of bits you want:"
read*, n
print*, "Enter the value of B:"
read*, B

! Calculate the number of states
num = 2**n
! Allocate arrays after determining the value of num
allocate(character(len=n) :: state(num))
allocate(integer :: decimal(num))
allocate(integer :: psi(n))
allocate(real(8) :: S1(num, num))
allocate(real(8) :: S2(num, num))
allocate(real(8) :: H(num, num))
allocate(real(8) :: H1(num, num))

! LAPACK parameters
lda = num
lwork = 3 * num - 1
allocate(real(8) :: w(lda), work(lwork))
```

```
do i = 0, num - 1
    call integer_binary(i, state(i + 1), n)  ! Adjusting index for
   1-based
    !print *, state(i + 1)
end do

! Convert binary states to decimal
do i = 1, num
    decimal(i) = btod(state(i), n)
end do

! Perform calculations based on the decimal values for S1
do i = 1, num
    do j = 1, num
        call single_digit(state(i), psi)
        if (decimal(i) /= decimal(j)) then
            S1(i, j) = 0.0
        else
            S1(i, j) = 0.0
            do k = 1, n - 1
                S1(i, j) = S1(i, j) + (1.0 - 2.0 * psi(k)) * (1.0 -
   2.0 * psi(k + 1))
            end do
            S1(i, j) = S1(i, j) + (1.0 - 2.0 * psi(n)) * (1.0 - 2.0
   * psi(1))
        end if
    end do
end do

! Construct the S2 matrix (for sigma_x )
S2 = 0.0
do i = 1, num
    do j = 1, num
        ! Initialize differing_bits counter
        differing_bits = 0
        do k = 1, n
            ! Compare the k-th bit of the i-th and j-th states
            if (btest(decimal(i), k-1) .neqv. btest(decimal(j), k-1)
```

```
    ) then
73                  differing_bits = differing_bits + 1
74              end if
75          end do
76          ! If exactly one bit differs, set S2(i, j) = 1.0
77          if (differing_bits == 1) then
78              S2(i, j) = 1.0
79          else
80              S2(i, j) = 0.0
81          end if
82      end do
83  end do
84
85  ! Total Hamiltonian
86  H =  S1 +  S2
87  ! Eigenvalue and eigenvector determination using DSYEV
88  JOBZ = 'V'
89  UPLO = 'U'
90  call DSYEV(JOBZ, UPLO, lda, H, lda, w, work, lwork, info)  if (info
    == 0) then
91      write(722,*) "Eigenvalue"
92      do a = 1, lda
93          write(722,*) 'W=(',a,')',w(a)
94      end do
95  else
96      write(722,*) 'Error in DSYEV, INFO=', info
97  end if
98  contains
99      function btod(state, n) result(decimal)
100         implicit none
101         integer, intent(in) :: n
102         character(len=n) :: state
103         integer :: decimal, i
104         decimal = 0
105         do i = 1, n
106             if (state(i:i) == '1') then
107                 decimal = decimal + 2**(n-i)
108             end if
```

```
109         end do
110     end function btod
111 
112     subroutine integer_binary(num, binary_string, n)
113         implicit none
114         integer :: num
115         integer, intent(in) :: n
116         integer :: i, temp_num
117         character(len=n), intent(out) :: binary_string
118         temp_num = num
119         binary_string = repeat('0', n)
120         do i = 0, n - 1
121             if (mod(temp_num, 2) == 1) then
122                 binary_string(n-i:n-i) = '1'
123             else
124                 binary_string(n-i:n-i) = '0'
125             end if
126             temp_num = temp_num / 2
127         end do
128     end subroutine integer_binary
129 
130     subroutine single_digit(A, psi)
131         implicit none
132         character(len=*), intent(in) :: A
133         integer, intent(out) :: psi(:)
134         integer :: i
135         do i = 1, len(A)
136             if (A(i:i) == '0') then
137                 psi(i) = 0
138             else if (A(i:i) == '1') then
139                 psi(i) = 1
140             else
141                 print *, "Invalid input in single_digit subroutine"
142             end if
143         end do
144     end subroutine single_digit
145 end program Ising_model_for_n_number_of_qubit
```

Let us see now how these energy eigenvalues are varied with the change in interaction energies and with the change in magnetic field. For doing so, we can adopt the adjacent flowchart Fig.[1.3] (where it is shown that J(interaction energy) and B(magnetic field) are varied from 0 to 1 in the interval of 0.01 unit).

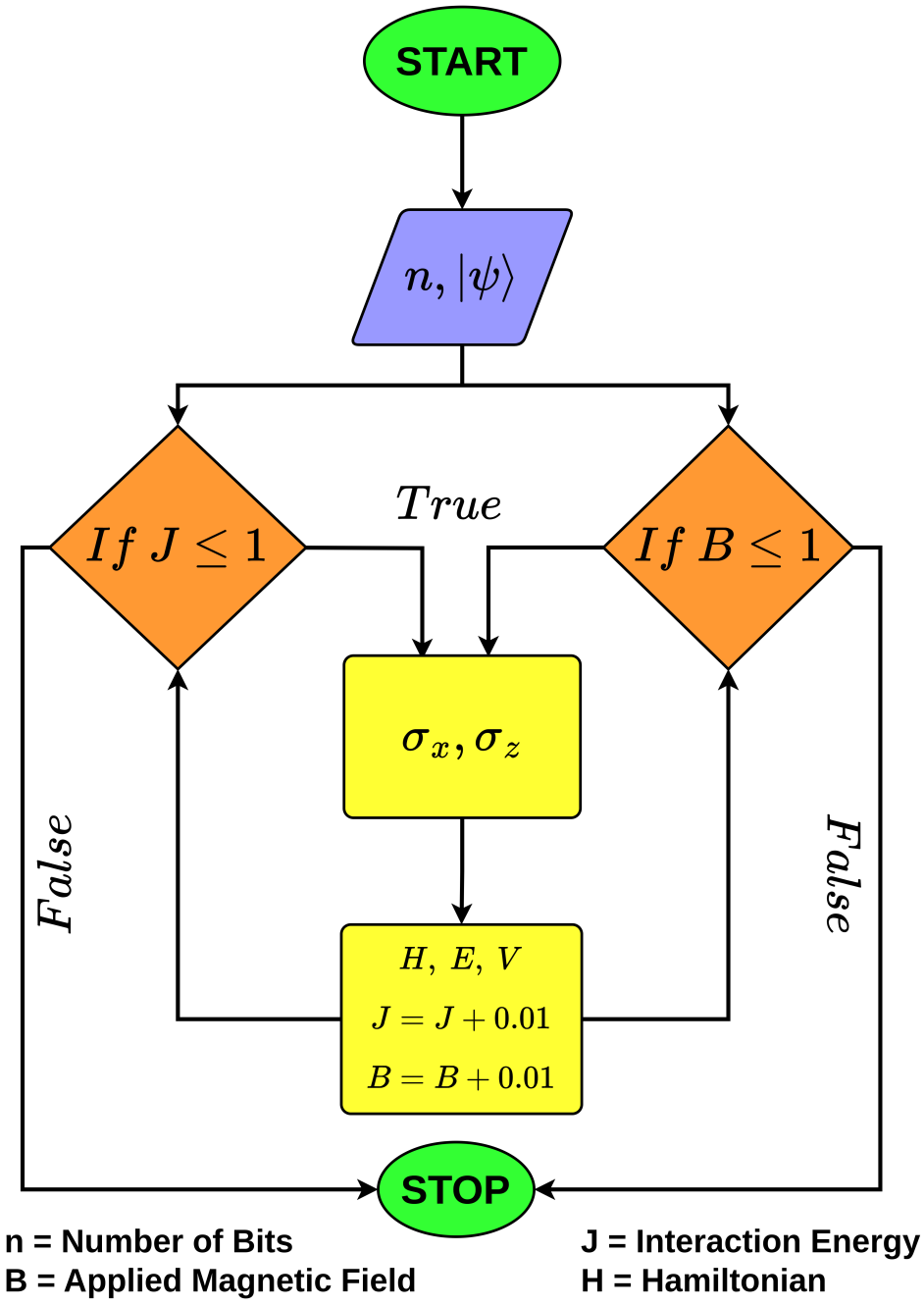


Figure 1.3: Flowchart illustrating the variation of energy eigenvalues with interaction energies and magnetic field.

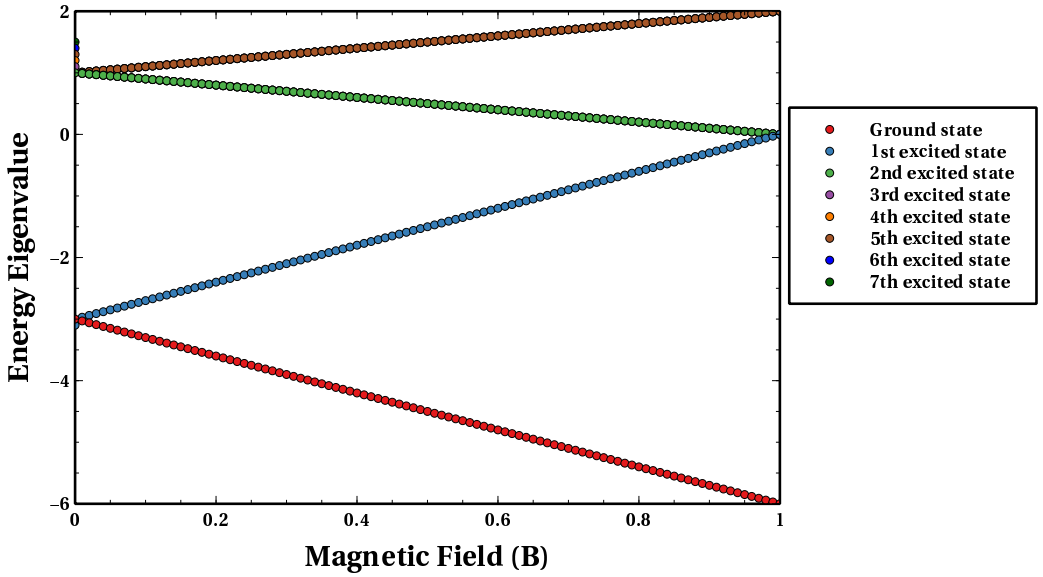


Figure 1.4: Variation of energy eigenvalue with magnetic field in the one-dimensional Ising model for case(1).

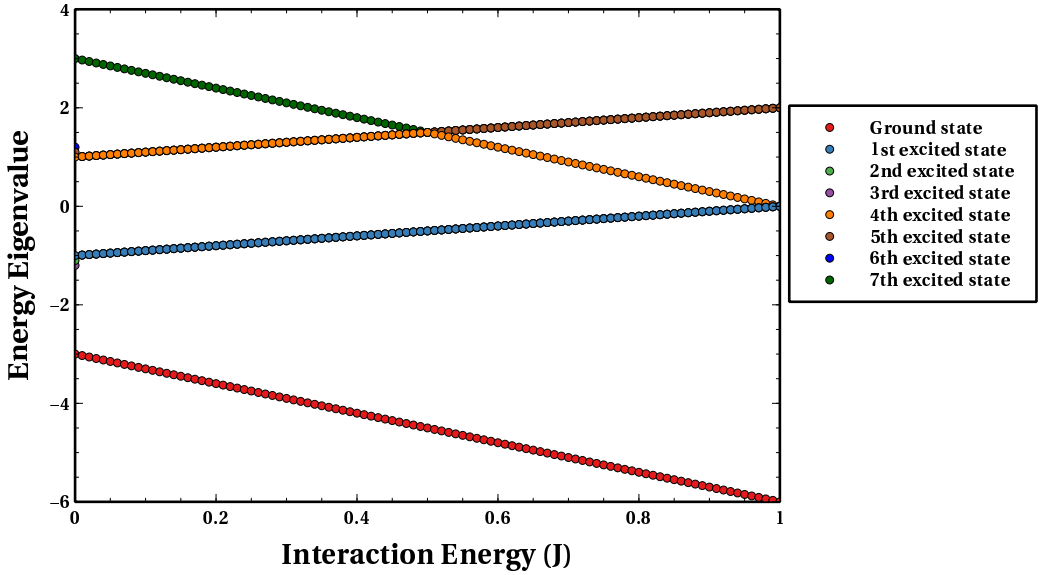


Figure 1.5: Variation of energy eigenvalue with interaction energy in the one-dimensional Ising model for case(1).

For case (1), we can see from the Fig[1.4] and Fig.[1.5] that we are getting a straight line-like variation of the eigenvalues for varying both the magnetic field and interaction energies separately. This is kind of obvious because the Hamiltonian contains only the σ^z terms, which is already in the diagonal basis. Now, σ_z and σ_x

do not commute with each other [4]. So, a common set of eigenbasis is not possible for the transverse Ising-like systems. Therefore, we shall not be able to get such a straight line-like variation of eigenvalues with both interaction energies as well as the magnetic field, as can be seen below in Fig.[1.6] and Fig.[1.7]

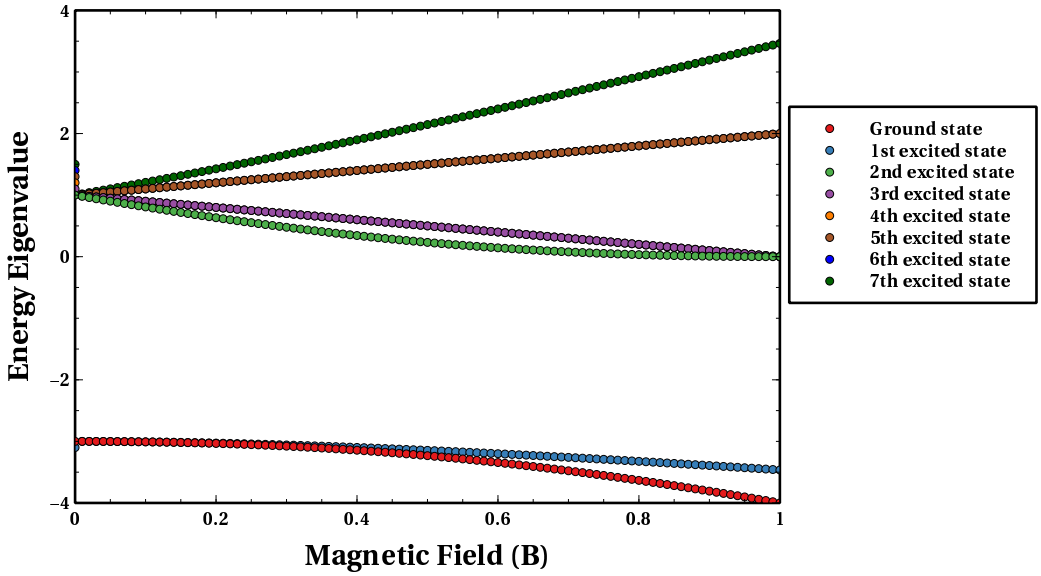


Figure 1.6: Variation of energy eigenvalues with magnetic field in the one-dimensional Ising model for a 3-qubit system considering case(2).

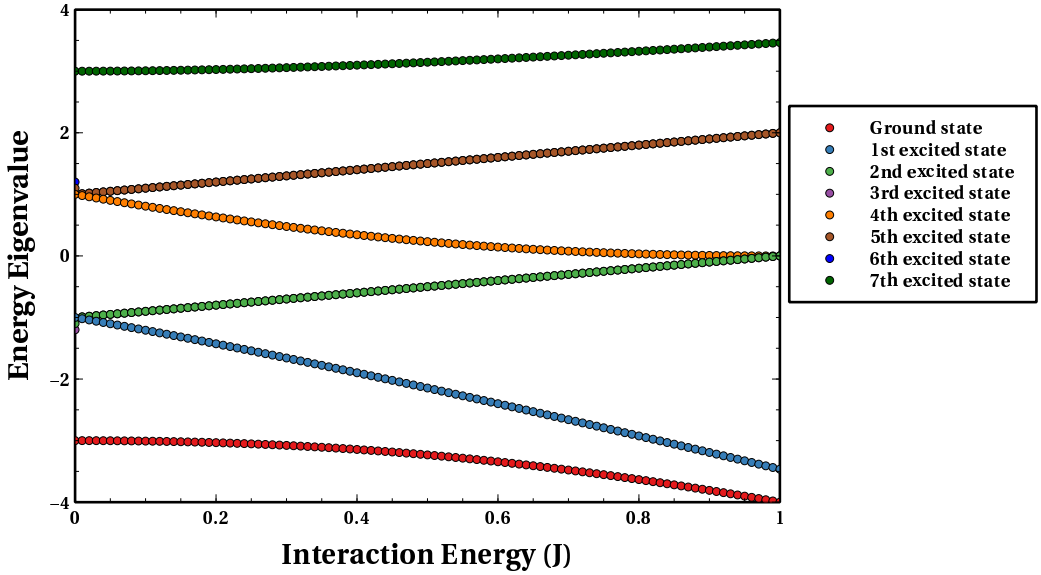


Figure 1.7: Variation of energy eigenvalues with interaction energy in the one-dimensional Ising model for 3 qubit system considering case(2).

1.2 Introduction to One-Dimensional Heisenberg Model

In this section, we will discuss one-dimensional Heisenberg's model of spin chain, which is generally used to study a magnetic system's critical points and phase transitions. It is a more realistic model where the spins corresponding to the magnetic systems are treated quantum mechanically. The Hamiltonian corresponding to the nearest neighbor one-dimensional Heisenberg model for Ninteracting spin $\frac{1}{2}$ particles (here we are considering qubits) can be given as follows [5] [6] [7] [8] [9] [10].

$$\mathcal{H} = \sum_{i=1}^{N} J_i \vec{\sigma_i}.\vec{\sigma}_{i+1} \quad . \tag{1.11}$$

It may come to your mind how the Hamiltonian has been constructed. Let us discuss this in detail. For this, we can start with a simple model for the covalent bonding of homonuclear molecules (like the H_2 molecule). We have considered such a homonuclear molecule as shown below in Fig.[1.8].

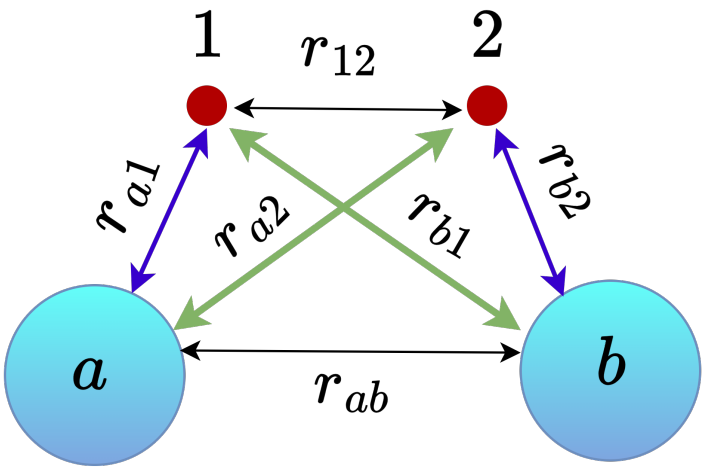


Figure 1.8: Homonuclear molecule consisting of two one-electron atoms

In this figure, we considered a homonuclear molecule consisting of two one-electron

atoms (we have taken the example of the H_2 molecule). The nuclei of the two atoms are indicated by a & b and that of the two electrons as 1 and 2, respectively. The distances are defined as follows: r_{ab} is the distance between the two nuclei, r_{12} is the distance between the two electrons, r_{a1} and r_{b1} are the distances between electron 1 and nuclei a and b respectively and r_{a2} and r_{b2} are the distances between electron 2 and nuclei a and b respectively. The masses of the two electrons are the same (i.e. $m_1 = m_2$), and the total Hamiltonian corresponding to the molecule can be written as,

$$\mathcal{H} = \underbrace{\frac{p_1^2}{2m_1} + \frac{p_2^2}{2m_2} - \frac{e^2}{4\pi\epsilon_0}\left(\frac{1}{r_{a1}} + \frac{1}{r_{b2}}\right)}_{\mathcal{H}_0 \text{ (Unperturbed Hamiltonian)}} \underbrace{- \frac{e^2}{4\pi\epsilon_0}\left(\frac{1}{r_{a2}} + \frac{1}{r_{b1}}\right) + \frac{e^2}{4\pi\epsilon_0}\left(\frac{1}{r_{12}} + \frac{1}{r_{ab}}\right)}_{\mathcal{H}_{\text{pert}} \text{ (Perturbed Hamiltonian)}} .$$

The total wave function of the system can be represented by the product of the two individual wave functions of the electrons. The total wave function contains one spin part and one space part, such that it must be antisymmetric (to satisfy Pauli's exclusion principle). Now, there are two possibilities for the spin configuration of the electrons, which are as follows in Fig.[1.9].

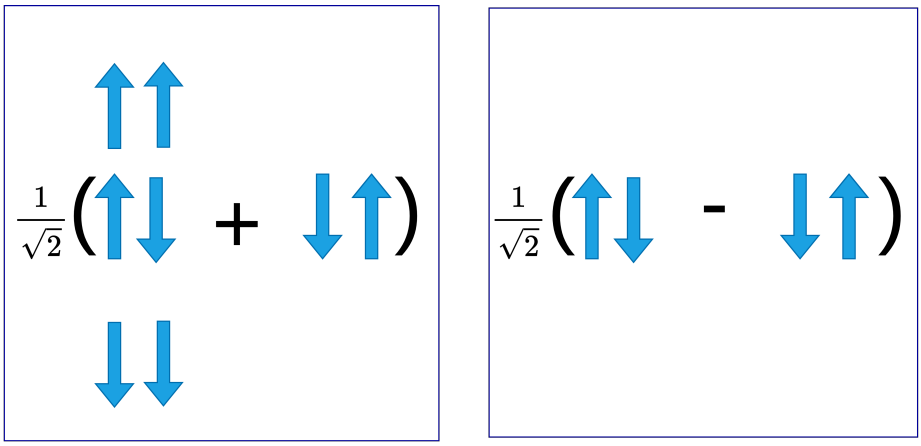


Figure 1.9: Spin Configuration for (a) Symmetric and (b) Antisymmetric arrangement

The total spin state corresponding to Fig.[1.9(a)], i.e., symmetric arrangement, is $S = 1$ (triplet), which makes the space part antisymmetric. Similarly, in the

antisymmetric arrangement like Fig.[1.9(b)], the total spin state corresponding to the two electrons is $S = 0$ (singlet), but the space part must be symmetric for this case. Mathematically, let $\psi_a(1)\psi_b(2)$ represents the 1st electron of 'a' atom and the 2nd electron of corresponding to 'b' atom, and $\psi_a(2)\psi_b(1)$ represents the 2nd electron of 'a' atom and the 1st electron of corresponding to 'b' atom. As the two electrons are identical particles, the total state of the system can be written as:

$$\Psi_s = \frac{1}{\sqrt{2}}(\psi_a(1)\psi_b(2) + \psi_a(2)\psi_b(1))\eta_s \tag{1.12}$$

$$\Psi_T = \frac{1}{\sqrt{2}}(\psi_a(1)\psi_b(2) - \psi_a(2)\psi_b(1))\eta_T \tag{1.13}$$

corresponding to singlet and triplet states, respectively, where η_s and η_T are the spin part of the state for singlet and triplet states, respectively.

The energy associated with these states can be expressed quantum mechanically as,

$$E_S = \int \Psi_S^* \mathcal{H} \Psi_S \, \mathrm{d}\mathbf{r}_1 \, \mathrm{d}\mathbf{r}_2, \tag{1.14}$$

$$E_T = \int \Psi_T^* \mathcal{H} \Psi_T \, \mathrm{d}\mathbf{r}_1 \, \mathrm{d}\mathbf{r}_2. \tag{1.15}$$

The energy difference between these states can be given by,

$$\begin{aligned}
E_S - E_T = \frac{1}{2}\int &([(\psi_a^*(1)\psi_b^*(2) + \psi_a^*(2)\psi_b^*(1))\mathcal{H}(\psi_a(1)\psi_b(2) + \psi_a(2)\psi_b(1))] - \\
&[(\psi_a^*(1)\psi_b^*(2) - \psi_a^*(2)\psi_b^*(1))\mathcal{H}(\psi_a(1)\psi_b(2) - \psi_a(2)\psi_b(1))])\, \mathrm{d}\mathbf{r}_1 \, \mathrm{d}\mathbf{r}_2, \\
= \frac{1}{2}\int &[(\psi_a^*(1)\psi_b^*(2))\mathcal{H}(\psi_a(1)\psi_b(2)) + (\psi_a^*(1)\psi_b^*(2))\mathcal{H}(\psi_a(2)\psi_b(1)) \\
&+ (\psi_a^*(2)\psi_b^*(1))\mathcal{H}(\psi_a(1)\psi_b(2)) + (\psi_a^*(2)\psi_b^*(1))\mathcal{H}(\psi_a(2)\psi_b(1)) \\
&- (\psi_a^*(1)\psi_b^*(2))\mathcal{H}(\psi_a(1)\psi_b(2)) + (\psi_a^*(1)\psi_b^*(2))\mathcal{H}(\psi_a(2)\psi_b(1)) \\
&+ (\psi_a^*(2)\psi_b^*(1))\mathcal{H}(\psi_a(1)\psi_b(2)) - (\psi_a^*(2)\psi_b^*(1))\mathcal{H}(\psi_a(2)\psi_b(1))]\, \mathrm{d}\mathbf{r}_1 \, \mathrm{d}\mathbf{r}_2, \\
= \frac{1}{2}\int &[2(\psi_a^*(1)\psi_b^*(2))\mathcal{H}(\psi_a(2)\psi_b(1)) + 2(\psi_a^*(2)\psi_b^*(1))\mathcal{H}(\psi_a(1)\psi_b(2))]\, \mathrm{d}\mathbf{r}_1 \, \mathrm{d}\mathbf{r}_2 \\
= \frac{1}{2}\int &4[(\psi_a^*(1)\psi_b^*(2))\mathcal{H}(\psi_a(2)\psi_b(1))]\, \mathrm{d}\mathbf{r}_1 \, \mathrm{d}\mathbf{r}_2,
\end{aligned}$$

because of the Hermitian property of $\mathcal{H}$, i.e.,

$$\int [(\psi_a^*(2)\psi_b^*(1))\mathcal{H}(\psi_a(1)\psi_b(2))] \mathrm{d}\mathbf{r}_1 \, \mathrm{d}\mathbf{r}_2 = \int [(\psi_a^*(1)\psi_b^*(2))\mathcal{H}(\psi_a(2)\psi_b(1))] \mathrm{d}\mathbf{r}_1 \, \mathrm{d}\mathbf{r}_2$$

$$E_S - E_T = 2\int [(\psi_a^*(1)\psi_b^*(2))\mathcal{H}(\psi_a(2)\psi_b(1))]\,\mathrm{d}\mathbf{r}_1\,\mathrm{d}\mathbf{r}_2.$$

Therefore,

$$\boxed{E_S - E_T = 2\int \psi_a^*(1)\psi_b^*(2)\mathcal{H}\psi_a(2)\psi_b(1)\,\mathrm{d}\mathbf{r}_1\,\mathrm{d}\mathbf{r}_2} \tag{1.16}$$

where, $\mathbf{r}_1$ and $\mathbf{r}_2$ are the position vectors of the two electrons. Hence, the total Hamiltonian of the system in terms of the 'Effective Hamiltonian' can be written as follows.

$$\mathcal{H} = \frac{1}{4}(E_s + 3E_T) - (E_s - E_T)\vec{s_1}.\vec{s_2} \tag{1.17}$$

The first term is constant, and we are more interested in the second term, which depends on the spin. Now it would be simpler to model the two-spin system by a spin Hamiltonian which contains only a spin operator in such a way as to bring out these essential details of the energy level scheme with the triplet state lying below the singlet state by an amount $2J'$ for $J' > 0$. This type of Hamiltonian model was first proposed by Heisenberg, and it was given as,

$$\mathcal{H}_{\mathbf{spin}} = -2J'\vec{s_1}.\vec{s_2} \tag{1.18}$$

where J' is the exchange integral given by,

$$J' = \int \psi_a^*(1)\psi_b^*(2)\mathcal{H}\psi_a(2)\psi_b(1)\,\mathrm{d}\mathbf{r}_1\,\mathrm{d}\mathbf{r}_2, \text{ (from Eq.[1.16])} \tag{1.19}$$

By applying this Hamiltonian from Eq.[1.18], we can easily get the difference between the energy levels of the singlet and triplet state as $2J'$. The steps are shown below. If each electron has a spin angular momentum quantum number of half (i.e., $s = \frac{1}{2}$), this leads to a total spin angular momentum quantum number,

$$S = \begin{cases} 0 & \text{(singlet)} \\ 1 & \text{(triplet)} \end{cases}$$

We know,

$$\vec{s_1} \cdot \vec{s_2} = \frac{(S^2 - s_1^2 - s_2^2)}{2} \tag{1.20}$$

For the triplet case ($S = 1$),

$$\vec{s_1} \cdot \vec{s_2} = \left[\frac{1(1+1) - \frac{1}{2}(1+\frac{1}{2}) - \frac{1}{2}(1+\frac{1}{2})}{2}\right]\hbar^2 = \frac{1}{4}\hbar^2 \tag{1.21}$$

For the singlet case ($S = 0$),

$$\vec{s_1} \cdot \vec{s_2} = \left[\frac{0(0+1) - \frac{1}{2}(1+\frac{1}{2}) - \frac{1}{2}(1+\frac{1}{2})}{2} \right] \hbar^2 = -\frac{3}{4}\hbar^2 \tag{1.22}$$

Again in terms of the Pauli spin matrix we can write $\vec{s_1} = \frac{\hbar}{2}\vec{\sigma_1}$ and $\vec{s_2} = \frac{\hbar}{2}\vec{\sigma_2}$. This gives us,

$$\vec{\sigma_1} \cdot \vec{\sigma_2} = \frac{4}{\hbar^2}\vec{s_1} \cdot \vec{s_2} = \frac{4}{\hbar^2}.\frac{1}{4}\hbar^2 = 1, \qquad \text{(for triplet)} \tag{1.23}$$

$$\vec{\sigma_1} \cdot \vec{\sigma_2} = \frac{4}{\hbar^2}\vec{s_1} \cdot \vec{s_2} = -\frac{4}{\hbar^2}.\frac{3}{4}\hbar^2 = -3, \qquad \text{(for singlet)} \tag{1.24}$$

Therefore, the interaction energies for triplet and singlet states are

$$\mathbf{E_T} = -\frac{J'}{2}, \quad \mathbf{E_S} = \frac{3J'}{2} \tag{1.25}$$

yielding the difference as

$$\mathbf{E_S} - \mathbf{E_t} = 2J' \tag{1.26}$$

Now, in a system of several spins, the Hamiltonian can be written for a pair of electron spins,

$$\mathcal{H}_{\text{spin}} = -\sum_{j>i} J'_{ij}\vec{s_i} \cdot \vec{s_j} = -\sum_{j>i} J_{ij}\vec{\sigma}_i \cdot \vec{\sigma}_j$$

Here, the Hamiltonian is written in terms of the Pauli spin matrices, where

$$J_{ij} = J'_{ij}\frac{4}{\hbar^2}$$

is the exchange interaction energy between the i^{th} and j^{th} spins, with the condition $j > i$ to avoid double counting, and there is no self-interaction also.

We are going to solve and analyze the eigenvalue spectrum corresponding to the one-dimensional Heisenberg model in Chapter 5.

1.3 Chapter Summary

In this chapter, we have discussed some background regarding the one-dimensional Ising model and the one-dimensional Heisenberg model to give a flavour of spin

chains and which will be of help for the reader. We also discussed the numerical solution scheme for the Hamiltonian corresponding to one-dimensional Ising mode by two methods, the first one is a pretty straightforward implementation of tensor products, and the second one involves the bit testing approach, which is a smarter way to construct the spin chain Hamiltonian corresponding to many qubits. We also discussed the energy eigenvalue spectrum corresponding to the variation of interaction energies and the magnetic field separately for two systems- the first one for the isotropic Ising model and the second one for the transverse Ising model. We also analyzed the eigenvalue spectrum numerically with respect to interaction energy (J) and magnetic field (B). The eigenvalue spectrum corresponding to the Heisenberg model is discussed in Chapter 5.

Chapter 2

Fibonacci sequence and Its Importance

2.1 Introduction

The Fibonacci sequence was introduced by the Italian mathematician Leonardo Pisano (whose nickname was Fibonacci) in his book *Liber Abaci*[11], which originally modeled the growth of a rabbit population under idealized conditions [12]. In his thought experiment, Fibonacci assumed that if a pair of newborn rabbits be placed in some place then how many rabbits would be there after one year considering certain assumptions that the pair of rabbits always produces one male and one female offspring, the rabbits reproduce once every year, they can reproduce once they are one month old, they never die.

Following these assumptions, one pair of rabbits will only be there after one month because they are too young to reproduce. At the end of the second month, the pair becomes old enough to reproduce. Then they produce a pair of newborn offspring in the next month, so that the total number of rabbit pairs becomes 2. In the fourth month, the infant offspring are too young to reproduce, but the first pair can reproduce, and another newborn pair of offspring is produced, so that the total number of rabbit pairs becomes 3. Now the second pair also reproduces like the

first pair, so that the first two pairs can reproduce another new offspring pair, but the latest pair is not old enough to reproduce, so the total number of rabbit pairs becomes 5. Likewise, it will continue. Now, Fibonacci was worried about the total number of rabbit pairs at the end of the year, and following the sequence, he was able to find the number to be 144. We can see this thing in tabular form as follows,

Month	1	2	3	4	5	6	7	8	9	10	11	12
Number of pairs	1	1	2	3	5	8	13	21	34	55	89	144

Table 2.1: Number of rabbit pairs showing the Fibonacci sequence.

2.2 Fibonacci Sequence and Golden Ratio

From the result of the above thought experiment, Fibonacci observed an interesting recursive relation or sequence of numbers where each number is the sum of the last two preceding terms in the sequence. This is known as the Fibonacci sequence or Fibonacci series. Mathematically, the series can be given by the recursive relation as follows,

$$F_{n+1} = F_n + F_{n-1}. \tag{2.1}$$

where, F_n is the n^{th} term of the Fibonacci sequence,F_{n-1} is the $(n-1)^{th}$ term and F_{n+1} is the $(n+1)^{th}$ term of that sequence. So if we start from 1 i.e. $F_0 = 1$ & $F_1 = 1$ the series will be like this,

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \ldots$$

Now we can generate the Fibonacci sequence and determine the terms with the help of Fortran as follows.

```
1 program Fibonacci_Series_generation
2     implicit none
3     integer :: n, i
4     double precision, allocatable :: fib(:)
5
6     ! Prompt the user for the number of terms
7     print*, "Enter the number of terms in the Fibonacci series:"
```

```
    read*, n
    if (n <= 0) then
        print*, "Number of terms must be greater than 0."
        stop
    end if

    ! Allocate fib
    allocate(fib(n))

    ! Initialize the first two terms
    fib(1) = 1
    if (n > 1) then
        fib(2) = 1
    end if

    ! Generate the Fibonacci series
    do i = 3, n
        fib(i) = fib(i-1) + fib(i-2)
    end do

    ! Print the Fibonacci series
    print*, "Fibonacci series:"
    do i = 1, n
        print*, fib(i)
    end do

    ! Deallocate arrays
    deallocate(fib)

contains
    subroutine sum(n, fib, total_sum)
        implicit none
        integer, intent(in) :: n
        double precision, intent(in) :: fib(:)
        double precision, intent(out) :: total_sum
        integer :: i

        total_sum = 0.0d0
```

```
        do i = 1, n
            total_sum = total_sum + fib(i)
        end do
    end subroutine sum

end program Fibonacci_Series_generation
```

Now, one interesting thing in the Fibonacci series that can be observed is the Golden ratio (ϕ), which is defined as the ratio of two successive numbers of the Fibonacci series when the number of terms is very large. It can be seen that at the start of the series, the ratio between the two successive terms is not constant; rather, it is used to vary, but after a certain number of terms, the ratio becomes constant, and the value of the constant becomes nearly equal to 1.618. Mathematically, the Golden ratio is obtained by,

$$\begin{aligned} F_{n+1} &= F_n + F_{n-1}, \\ \frac{F_{n+1}}{F_n} &= 1 + \frac{F_{n-1}}{F_n}, \\ \phi &= 1 + \frac{1}{\phi}, \\ \phi^2 - \phi - 1 &= 0, \\ \phi &= \frac{\sqrt{5}+1}{2}. \end{aligned} \tag{2.2}$$

Therefore, the Golden ratio, $\phi = \frac{\sqrt{5}+1}{2} \approx 1.618$.

2.2.1 Golden Ratio obtained for 100 numbers of the Fibonacci Series

We can show how the Golden ratio is achieved by taking the first 100 numbers of the Fibonacci series and calculating the ratio between successive terms. To do this, we can modify the previous code as follows.

```
program Fibonacci_Series_generation
    implicit none
    integer :: n, i
```

```
    double precision, allocatable :: fib(:),ratio

    ! Prompt the user for the number of terms
    print*, "Enter the number of terms in the Fibonacci series:"
    read*, n
    if (n <= 0) then
        print*, "Number of terms must be greater than 0."
        stop
    end if

    ! Allocate fib
    allocate(fib(n))

    ! Initialize the first two terms
    fib(1) = 1
    if (n > 1) then
        fib(2) = 1
    end if

    ! Generate the Fibonacci series
    do i = 3, n
        fib(i) = fib(i-1) + fib(i-2)
    end do

    ! Print the Fibonacci series
    print*, "Fibonacci series:"
    do i = 1, n
        print*, fib(i)
    end do

    ! Calculate the ratio between two successive terms of the
   Fibonacci series
    do i = 2, n
        if (fib(i) /= 0.0d0) then
            ratio = fib(i) / fib(i-1)
        end if
        print*, ratio
        write(100,*) i, ratio
```

```
    end do

    ! Deallocate arrays
    deallocate(fib)

contains
    subroutine sum(n, fib, total_sum)
        implicit none
        integer, intent(in) :: n
        double precision, intent(in) :: fib(:)
        double precision, intent(out) :: total_sum
        integer :: i

        total_sum = 0.0d0
        do i = 1, n
            total_sum = total_sum + fib(i)
        end do
    end subroutine sum

end program Fibonacci_Series_generation
```

The output is shown below. The first column shows the number of terms, and the second column shows the ratio between successive terms. From this output file, we can see that the ratio becomes equal to the golden ratio after 20 terms.

Table 2.2: Convergence of Ratio Between Successive Terms

Number of Terms	**Ratio Between Successive Terms**
1	1.0000000000000000
2	1.0000000000000000
3	2.0000000000000000
4	1.5000000000000000
5	1.6666666666666667
6	1.6000000000000001
7	1.6250000000000000
8	1.6153846153846154

Number of Terms	Ratio Between Successive Terms
9	1.6190476190476191
10	1.6176470588235294
11	1.6181818181818182
12	1.6179775280898876
13	1.6180555555555556
14	1.6180257510729614
15	1.6180371352785146
16	1.6180327868852460
17	1.6180344478216819
18	1.6180338134001253
19	1.6180340557275541
20	1.6180339631667064
21	1.6180339985218033
22	1.6180339850173580
23	1.6180339901755971
24	1.6180339882053250
25	1.6180339889579021
26	1.6180339886704431
27	1.6180339887802426
28	1.6180339887383031
29	1.6180339887543225
30	1.6180339887482036
31	1.6180339887505408
32	1.6180339887496482
33	1.6180339887499890
34	1.6180339887498589
35	1.6180339887499087
36	1.6180339887498896
37	1.6180339887498969
38	1.6180339887498940

Number of Terms	Ratio Between Successive Terms
39	1.6180339887498951
40	1.6180339887498947
41	1.6180339887498949
42	1.6180339887498949
43	1.6180339887498949
44	1.6180339887498949
45	1.6180339887498949
46	1.6180339887498949
47	1.6180339887498949
48	1.6180339887498949
49	1.6180339887498949
50	1.6180339887498949
51	1.6180339887498949
52	1.6180339887498949
53	1.6180339887498949
54	1.6180339887498949
55	1.6180339887498949
56	1.6180339887498949
57	1.6180339887498949
58	1.6180339887498949
59	1.6180339887498949
60	1.6180339887498949
61	1.6180339887498949
62	1.6180339887498949
63	1.6180339887498949
64	1.6180339887498949
65	1.6180339887498949
66	1.6180339887498949
67	1.6180339887498949
68	1.6180339887498949

Number of Terms	Ratio Between Successive Terms
69	1.6180339887498949
70	1.6180339887498949
71	1.6180339887498949
72	1.6180339887498949
73	1.6180339887498949
74	1.6180339887498949
75	1.6180339887498949
76	1.6180339887498949
77	1.6180339887498949
78	1.6180339887498949
79	1.6180339887498947
80	1.6180339887498949
81	1.6180339887498949
82	1.6180339887498947
83	1.6180339887498949
84	1.6180339887498949
85	1.6180339887498949
86	1.6180339887498949
87	1.6180339887498949
88	1.6180339887498949
89	1.6180339887498947
90	1.6180339887498949
91	1.6180339887498949
92	1.6180339887498949
93	1.6180339887498949
94	1.6180339887498949
95	1.6180339887498947
96	1.6180339887498949
97	1.6180339887498949
98	1.6180339887498949

Number of Terms	Ratio Between Successive Terms
99	1.6180339887498949
100	1.6180339887498949

If we plot the above data, then it will clarify our statement that we had discussed above.

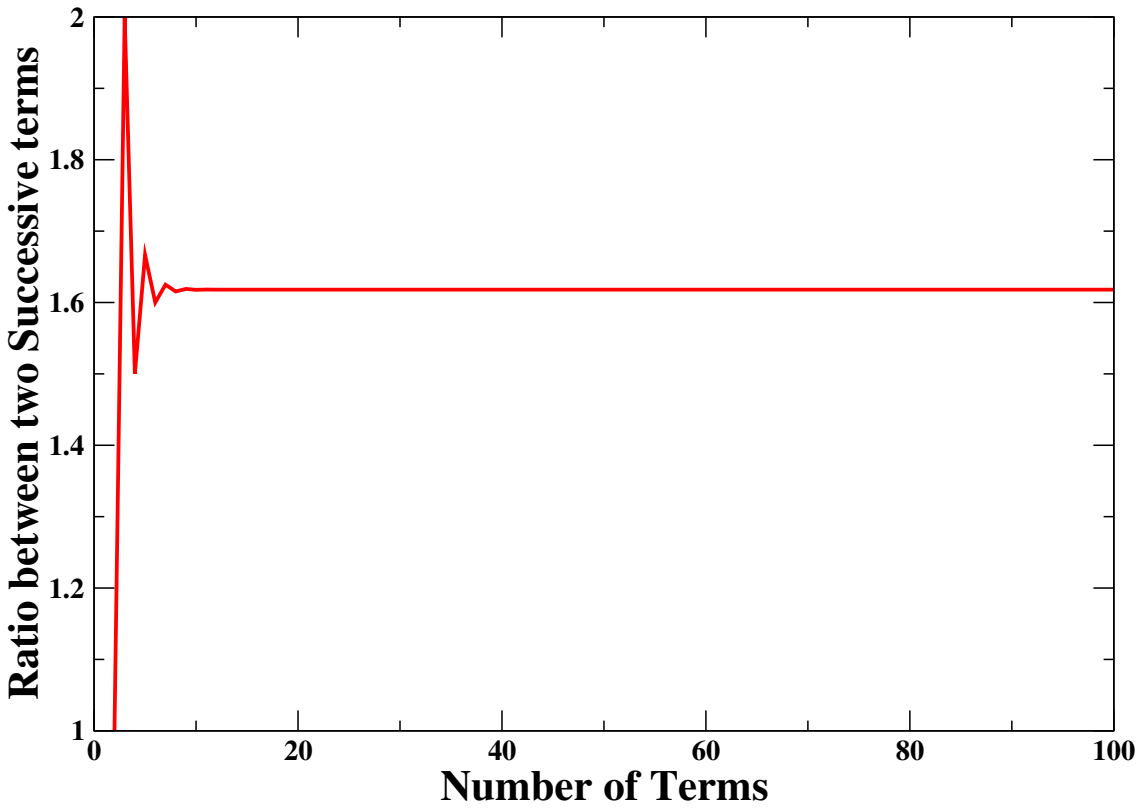


Figure 2.1: Golden ratio of Fibonacci series

2.3 Importance of Fibonacci Sequence

Fibonacci sequence is found in nature in many forms as one can find the Fibonacci type of sequence in the arrangement of petals of many flowers (like one petal for white callly lily, 3 petals for lily, iris, 5 petals for buttercup, wild rose etc., 8 petals for delphiniums, 13 petals for ragworts, corn marigold, cineria etc.)[12]. Also, we can observe the Fibonacci-like sequence in the arrangement of leaves of a tree. Nowadays

Fibonacci sequence and the golden ratio have found great interest in the field of research in science, including high-energy physics, quantum mechanics, etc. Because of its exciting properties, we can apply it in our linear one-dimensional spin system to explain some interesting phenomena, as can be described step by step when we go through the succeeding chapters.

2.4 Reciprocal Fibonacci series

The reciprocal Fibonacci series can easily be constructed by taking the reciprocal of the terms of the Fibonacci series. It is also very important in the context of physical problems. We can see that with the increase of the number of terms, the value of the Fibonacci number becomes very large and the series diverges. Whereas if we consider the reciprocal terms of the Fibonacci series, we shall be able to get a converging series, which is more important in the context of any physical system. The reciprocal Fibonacci series is given by,

$$\frac{1}{1}, \frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \frac{1}{5}, \frac{1}{8}, \frac{1}{13}, \frac{1}{21}, \frac{1}{34}, \frac{1}{55}, \frac{1}{89}, \frac{1}{144}, \ldots$$

Here also we can find the golden ratio for the reciprocal Fibonacci series as,

$$\psi = \frac{\frac{1}{F_{n+1}}}{\frac{1}{F_n}} \approx 0.618. \tag{2.3}$$

Now we can generate the reciprocal Fibonacci series by modifying our previous code as follows.

```
program Reciprocal_FibonacciSeries
    implicit none
    integer :: n, i
    double precision, allocatable :: fib(:)
    double precision, allocatable :: reciprocal(:)

    ! Prompt the user for the number of terms
    print*, "Enter the number of terms in the Fibonacci series:"
    read*, n
    if (n <= 0) then
        print*, "Number of terms must be greater than 0."
```

```
        stop
    end if

    ! Allocate fib and reciprocal arrays
    allocate(fib(n))
    allocate(reciprocal(n))

    ! Initialize the first two terms
    fib(1) = 1
    if (n > 1) then
        fib(2) = 1
    end if

    ! Generate the Fibonacci series
    do i = 3, n
        fib(i) = fib(i-1) + fib(i-2)
    end do

    ! Compute reciprocals
    do i = 1, n
        reciprocal(i) = 1.0d0 / real(fib(i))
    end do

    ! Deallocate arrays
    deallocate(fib)
    deallocate(reciprocal)

contains
    subroutine sum(n, fib, total_sum)
        implicit none
        integer, intent(in) :: n
        double precision, intent(in) :: fib(:)
        double precision, intent(out) :: total_sum
        integer :: i
        total_sum = 0.0d0
        do i = 1, n
            total_sum = total_sum + fib(i)
        end do
```

```
50      end subroutine sum
51  end program Reciprocal_FibonacciSeries
```

2.5 Chapter Summary

In this chapter, we discussed the history of Fibonacci numbers and also some interesting properties of them. We also discussed the Inverse Fibonacci series and generated it numerically. In the forthcoming chapters, we will show its application in our context.

★

Chapter 3

Transverse Ising-like model with the Inverse Fibonacci type of Interactions

---

3.1 Introduction

In the first chapter we saw the solution of one-dimensional Ising model and its eigenvalue spectrum for the variation of the magnetic field and spin interaction energies for the two cases - the first one in which both the spin interaction as well as the direction of applied magnetic field was in the same direction and the second one is when the magnetic field is applied to the transverse direction of the interaction energies. In the second chapter, we saw the Fibonacci sequence and its importance. In this chapter, we also considered the reciprocal Fibonacci sequence and said that for the physical reality, it is very important. In this chapter, we are going to see how we can construct a physical system (especially quasicrystal [15]) by using the reciprocal of the Fibonacci series in the one-dimensional Ising model [13] [14] and study the eigenvalue spectrum for this type of system.

3.1.1 What is Quasicrystal?

Solids are broadly classified into two classes, which are crystalline and amorphous solids. Crystalline solids possess an orderly arrangement of their basis (it may be an atom or a group of atoms), whereas amorphous solids do not possess such an ordered arrangement of their basis. Quasicrystals are something lying between these two classes- it does not show an orderly arrangement up to a certain distance, but they possess a long-range order in arrangement. Quasicrystals were first discovered in 1982 by Dan Shechtman while he was studying the molten mixture of aluminium (Al) and manganese (Mn). Some of the examples of quasicrystals are Al_6Cu_2Fe, $Al - Mn$ alloys, etc. Quasicrystals do not possess any translation symmetry, i.e., the unit cells are non-repetitive (as we can see from the Fig.[3.1]). The atomic structure of quasicrystals can often be considered using Penrose tiling [16] (shown below) or similar aperiodic slicing schemes. Quasicrystals find their application in surgical instruments, LED lights, nonstick frying pans, etc.

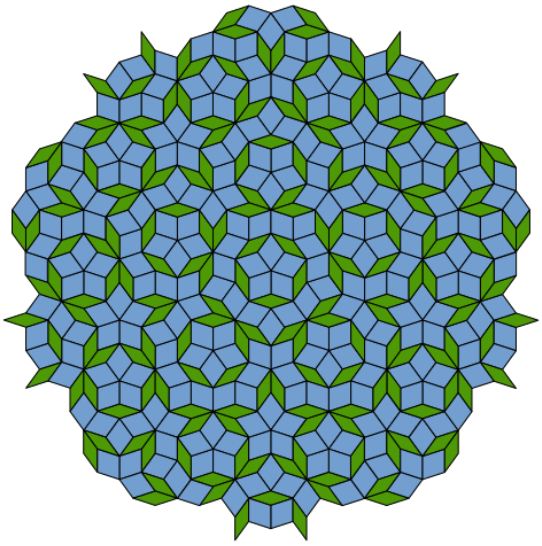


Figure 3.1: Penrose tiling [16]

3.1.2 Mathematical construction of Quasicrystal-like system

As said before, quasicrystals possess long-range order, which we can construct mathematically by a set of techniques, out of which the simplest one is by considering the Fibonacci series. We know that the Fibonacci series generates an aperiodic pattern which never repeats, analogous to the atomic arrangement of quasicrystals, and despite possessing an aperiodic nature Fibonacci series also possesses a long-range order, as well as mimicking the long-range order of quasicrystals. Again, the Fibonacci series converges to the golden ratio, observed many times in quasicrystal structures. Many tilings and atomic spacings in quasicrystals are related to the golden ratio. We are interested in the study of the entanglement between the spins present at the sites of a qusicrystal-like structure. For that purpose, it is assumed that the interaction energy between the neighbouring spins are the reciprocal of the numbers belonging to the Fibonacci series, or simply the inverse Fibonacci series. These are like random models where the interaction strengths between contiguous spins are not same but are chosen from the inverse Fibonacci series.

3.2 Transverse Ising-like model with the Inverse Fibonacci type of Interactions

Now, we can consider the Hamiltonian of the system corresponding to the transverse Ising-like model with inverse Fibonacci type of interactions (i.e. $J_i \in$ Inverse Fibonacci Series), which can be written as follows,

$$\mathcal{H} = \sum_i J_i \sigma_i^z \sigma_{i+1}^z + B \sum_i \sigma_i^x \quad . \tag{3.1}$$

Where we are considering the interaction between the neighboring spins to be along the Z-direction and the external magnetic field B applied along the X-direction, i.e., transverse to the spin chain system. Again, we have two cases to discuss as follows.

Case-I: Open Chain System

The Hamiltonian of the system for this case can be given by:

$$\mathcal{H} = \sum_{i=1}^{N-1} J_i \sigma_i^z \sigma_{i+1}^z + B \sum_{i=1}^{N} \sigma_i^x \tag{3.2}$$

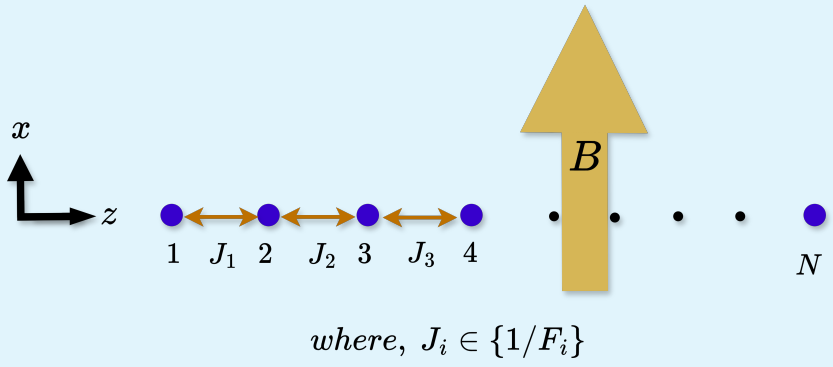


Figure 3.2: Open chain system

Case-II: Periodic Chain System

The Hamiltonian of the system for this case can be given by:

$$\mathcal{H} = \sum_{i=1}^{N} J_i \sigma_i^z \sigma_{i+1}^z + B \sum_{i=1}^{N} \sigma_i^x \tag{3.3}$$

where $\sigma_{N+1} = \sigma_1$.

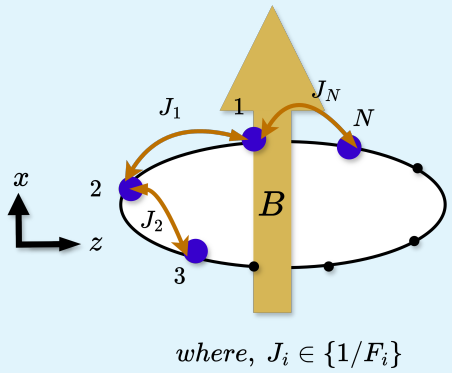


Figure 3.3: Periodic chain system

Let us solve these Hamiltonians respectively and find the variation of energy eigenvalues with respect to the change in the external magnetic field, which is varied

from 0 to 1 unit in steps of 0.01 unit for both cases (just as done in the first chapter for one one-dimensional Ising model).

3.2.1 Study of the Energy Eigenvalue Spectrum with the variation of Magnetic field

3.2.1.1 Case-(1) Open Chain System

We can study the energy eigenvalue spectrum corresponding to the change in the magnetic field (varied between 0 to 1 in steps of 0.01 unit) for the open chain system by constructing a Fortran code, which we have given below.

```
program Isinglike_model_with_inverse_Fibonacci_type_interaction
!Open chain System
!Here J is varied like the inverse of Fibonacci series and B is also
    varied from 0 to 1 unit in steps of 0.01 unit
implicit none
integer :: n, i, num, a, x, y, k, t, c, differing_bits
real*8 :: B
integer :: m
character(len=:), allocatable :: state(:)
integer, allocatable :: decimal(:)
integer, allocatable :: psi(:)
double precision, allocatable :: p(:)
integer :: j
real*8, allocatable :: S1(:,:), S2(:,:), H(:,:), H1(:,:), rho(:,:)

! Parameters for LAPACK DSYEV
integer :: lda, info, lwork
real*8, allocatable :: w(:), work(:)
character*1 :: JOBZ, UPLO

! Input values
print*, "Enter the number of bits you want:"
read*, n

```

```
! Calculate number of states
num = 2**n
m = n

! Allocate arrays after determining the value of num
allocate(character(len=n) :: state(num))
allocate(integer :: decimal(num))
allocate(psi(n))
allocate(S1(num, num))
allocate(S2(num, num))
allocate(H(num, num))
allocate(H1(num, num))
allocate(p(m))
allocate(rho(num,num))

! LAPACK parameters
lda = num
lwork = 3 * num - 1
allocate(real(8) :: w(lda), work(work))

! Convert binary states to decimal
do i = 1, num
    decimal(i) = btod(state(i), n)
    !write(722,*) decimal(i)
end do

call FibonacciSeries(m, p)
! Perform calculations based on the decimal values for S1
do i = 1, num
    do j = 1, num
        call single_digit(state(i), psi)
        if (decimal(i) /= decimal(j)) then
            S1(i, j) = 0.0
        else
            S1(i, j) = 0.0
             do k = 2, n
                S1(i, j) = S1(i, j) + p(k-1)*(1.0 - 2.0 * psi(k-1))
   * (1.0 - 2.0 * psi(k))
```

```
            end do
        end if
    end do
end do

! Construct the S2 matrix (for sigma_x )
S2 = 0.0
do i = 1, num
    do j = 1, num
        ! Initialize differing_bits counter
        differing_bits = 0
        do k = 1, n
        ! Compare the k-th bit of the i-th and j-th states
        if (btest(decimal(i), k-1) .neqv. btest(decimal(j), k-1))
   then
                differing_bits = differing_bits + 1
        end if
        end do
        ! If exactly one bit differs, set S2(i, j) = 1.0
        if (differing_bits == 1) then
            S2(i, j) = 1.0
        else
                S2(i, j) = 0.0
        end if
    end do
end do

! Total Hamiltonian
do y = 0, 100, 1
    B = y * 0.01
    H =  S1 + B * S2
    ! Eigenvalue and eigenvector determination using DSYEV
    JOBZ = 'V'
    UPLO = 'U'
    call DSYEV(JOBZ, UPLO, num, H, lda, w, work, lwork, info)
    if (info == 0) then
        do a = 1, lda
            write(1225,*) B, w(a) !plot for magfield vs eigenvalue
```

```
98          end do
99      else
100         write(1223,*) 'Error in DSYEV, INFO=', info
101     end if
102 end do
103
104 deallocate(state)
105 deallocate(decimal)
106 deallocate(psi)
107 deallocate(S1)
108 deallocate(S2)
109 deallocate(H)
110 deallocate(H1)
111 deallocate(p)
112 deallocate(rho)
113
114 contains
115 function btod(state, n) result(decimal)
116     implicit none
117     integer, intent(in) :: n
118     character(len=n) :: state
119     integer :: decimal, i
120     decimal = 0
121     do i = 1, n
122         if (state(i:i) == '1') then
123             decimal = decimal + 2**(n-i)
124         end if
125     end do
126 end function btod
127
128 subroutine integer_binary(num, binary_string, n)
129     implicit none
130     integer :: num
131     integer, intent(in) :: n
132     integer :: i, temp_num
133     character(len=n), intent(out) :: binary_string
134     temp_num = num
135     binary_string = repeat('0', n)
```

```
136     do i = 0, n - 1
137         if (mod(temp_num, 2) == 1) then
138             binary_string(n-i:n-i) = '1'
139         else
140             binary_string(n-i:n-i) = '0'
141         end if
142         temp_num = temp_num / 2
143     end do
144 end subroutine integer_binary
145 
146 subroutine single_digit(A, psi)
147     implicit none
148     character(len=*), intent(in) :: A
149     integer, intent(out) :: psi(:)
150     integer :: i
151     do i = 1, len(A)
152         if (A(i:i) == '0') then
153             psi(i) = 0
154         else if (A(i:i) == '1') then
155             psi(i) = 1
156         else
157             print *, "Invalid input in single_digit subroutine"
158         end if
159     end do
160 end subroutine single_digit
161 
162 subroutine FibonacciSeries(m, reciprocal)
163     implicit none
164     integer :: i
165     integer :: m
166     double precision :: fib(m)
167     double precision :: reciprocal(m)
168     if (m <= 0) then
169         print*, "Number of terms must be greater than 0."
170         stop
171     end if
172     ! Initialize the first two terms
173     fib(1) = 1
```

```
    if (n > 1) then
        fib(2) = 1
    end if
    ! Generate the Fibonacci series
    do i = 3, m
        fib(i) = fib(i-1) + fib(i-2)
    end do
    ! Compute reciprocals
    do i = 1, m
        reciprocal(i) = 1.0d0 / real(fib(i))
    end do
    !print*, fib
end subroutine FibonacciSeries
end program Isinglike_model_with_inverse_Fibonacci_type_interaction
```

We have constructed the above Fortran code applicable for systems corresponding to the N number of qubits, but for example, we can take a three-qubit system. The variation of the eigenvalues with magnetic field for the three-qubit open chain system is shown below.

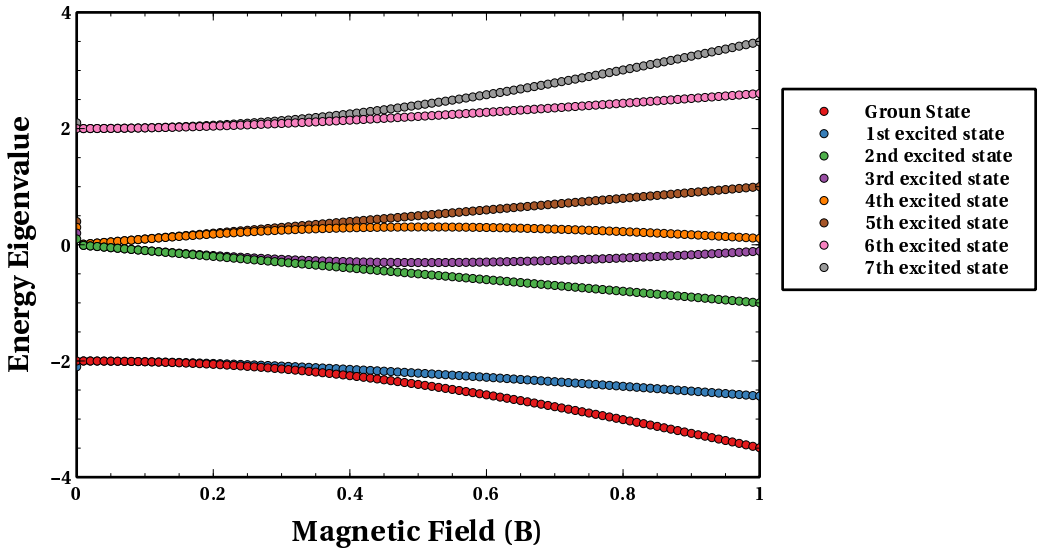


Figure 3.4: Variation of energy eigenvalue with Magnetic field for an open chain system consisting of three qubits when the interaction energies are taken along one direction.

The nature of the variation is not linear, which is quite obvious because we have taken interaction energies along one direction of the spin chain system and applied the magnetic field transverse to this system. So one single eigenfunction can not be able to diagonalise the complete Hamiltonian because the commutation of σ_z and σ_x is not zero, and so we can not be able to get an equation like a straight line as discussed already mathematically in Chapter 1.

3.2.1.2 Case-(2) Periodic Chain System

We can repeat the analysis in the case of a periodic chain system, for which we have to implement only the periodic condition in our previous code.

```
program Isinglike_model_with_inverse_Fibonacci_type_interaction
!For periodic chain system
!Here, J is varied like the inverse of the Fibonacci series, and B
   is also varied from 0 to 1 unit in steps of 0.01 unit
implicit none
integer :: n, i, num, a, x, y, k, t, c, differing_bits
real*8 :: B
integer :: m
character(len=:), allocatable :: state(:)
integer, allocatable :: decimal(:)
integer, allocatable :: psi(:)
double precision, allocatable :: p(:)
integer :: j
real*8, allocatable :: S1(:,:), S2(:,:), H(:,:), H1(:,:), rho(:,:)

! Parameters for LAPACK DSYEV
integer :: lda, info, lwork
real*8, allocatable :: w(:), work(:)
character*1 :: JOBZ, UPLO

! Input values
print*, "Enter the number of bits you want:"
read*, n

```

```
! Calculate number of states
num = 2**n
m = n

! Allocate arrays after determining the value of num
allocate(character(len=n) :: state(num))
allocate(integer :: decimal(num))
allocate(psi(n))
allocate(S1(num, num))
allocate(S2(num, num))
allocate(H(num, num))
allocate(H1(num, num))
allocate(p(m))
allocate(rho(num,num))

! LAPACK parameters
lda = num
lwork = 3 * num - 1
allocate(real(8) :: w(lda), work(work))

! Convert binary states to decimal
do i = 1, num
    decimal(i) = btod(state(i), n)
    !write(722,*) decimal(i)
end do

call FibonacciSeries(m, p)
! Perform calculations based on the decimal values for S1
do i = 1, num
    do j = 1, num
        call single_digit(state(i), psi)
        if (decimal(i) /= decimal(j)) then
            S1(i, j) = 0.0
        else
            S1(i, j) = 0.0
            do k = 1, n - 1
                S1(i, j) = S1(i, j) + p(k)*(1.0 - 2.0 * psi(k)) *
   (1.0 - 2.0 * psi(k + 1))
```

```
61              end do
62              S1(i, j) = S1(i, j) + p(n)*(1.0 - 2.0 * psi(n)) * (1.0 -
    2.0 * psi(1))
63          end if
64      end do
65  end do
66
67  ! Construct the S2 matrix (for sigma_x )
68  S2 = 0.0
69  do i = 1, num
70      do j = 1, num
71          ! Initialize differing_bits counter
72          differing_bits = 0
73          do k = 1, n
74          ! Compare the k-th bit of the i-th and j-th states
75          if (btest(decimal(i), k-1) .neqv. btest(decimal(j), k-1))
    then
76                  differing_bits = differing_bits + 1
77          end if
78          end do
79          ! If exactly one bit differs, set S2(i, j) = 1.0
80          if (differing_bits == 1) then
81              S2(i, j) = 1.0
82          else
83                  S2(i, j) = 0.0
84          end if
85      end do
86  end do
87
88  ! Total Hamiltonian
89  do y = 0, 100, 1
90      B = y * 0.01
91      H =  S1 + B * S2
92      ! Eigenvalue and eigenvector determination using DSYEV
93      JOBZ = 'V'
94      UPLO = 'U'
95      call DSYEV(JOBZ, UPLO, num, H, lda, w, work, lwork, info)
96      if (info == 0) then
```

```
97          do a = 1, lda
98              write(1225,*) B, w(a) !plot for magfield vs eigenvalue
99          end do
100     else
101         write(1223,*) 'Error in DSYEV, INFO=', info
102     end if
103 end do
104 
105 deallocate(state)
106 deallocate(decimal)
107 deallocate(psi)
108 deallocate(S1)
109 deallocate(S2)
110 deallocate(H)
111 deallocate(H1)
112 deallocate(p)
113 deallocate(rho)
114 
115 contains
116 function btod(state, n) result(decimal)
117     implicit none
118     integer, intent(in) :: n
119     character(len=n) :: state
120     integer :: decimal, i
121     decimal = 0
122     do i = 1, n
123         if (state(i:i) == '1') then
124             decimal = decimal + 2**(n-i)
125         end if
126     end do
127 end function btod
128 
129 subroutine integer_binary(num, binary_string, n)
130     implicit none
131     integer :: num
132     integer, intent(in) :: n
133     integer :: i, temp_num
134     character(len=n), intent(out) :: binary_string
```

```
135     temp_num = num
136     binary_string = repeat('0', n)
137     do i = 0, n - 1
138         if (mod(temp_num, 2) == 1) then
139             binary_string(n-i:n-i) = '1'
140         else
141             binary_string(n-i:n-i) = '0'
142         end if
143         temp_num = temp_num / 2
144     end do
145 end subroutine integer_binary
146 
147 subroutine single_digit(A, psi)
148     implicit none
149     character(len=*), intent(in) :: A
150     integer, intent(out) :: psi(:)
151     integer :: i
152     do i = 1, len(A)
153         if (A(i:i) == '0') then
154             psi(i) = 0
155         else if (A(i:i) == '1') then
156             psi(i) = 1
157         else
158             print *, "Invalid input in single_digit subroutine"
159         end if
160     end do
161 end subroutine single_digit
162 
163 subroutine FibonacciSeries(m, reciprocal)
164     implicit none
165     integer :: i
166     integer :: m
167     double precision :: fib(m)
168     double precision :: reciprocal(m)
169     if (m <= 0) then
170         print*, "Number of terms must be greater than 0."
171         stop
172     end if
```

```
    ! Initialize the first two terms
    fib(1) = 1
    if (n > 1) then
        fib(2) = 1
    end if
    ! Generate the Fibonacci series
    do i = 3, m
        fib(i) = fib(i-1) + fib(i-2)
    end do
    ! Compute reciprocals
    do i = 1, m
        reciprocal(i) = 1.0d0 / real(fib(i))
    end do
    !print*, fib
end subroutine FibonacciSeries
end program Isinglike_model_with_inverse_Fibonacci_type_interaction
```

The variation of the eigenvalues with magnetic field for the three-qubit periodic chain system is shown below in Fig[3.5].

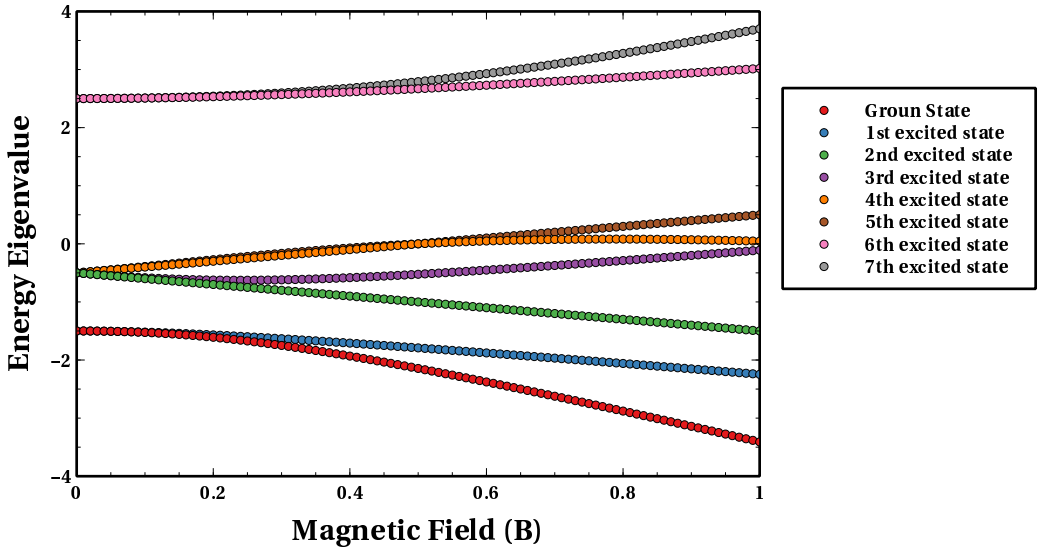


Figure 3.5: Variation of energy eigenvalue with magnetic field for a periodic chain system consisting of three qubits when the interaction energies are taken along one direction

The nature of the variation is similar to that of case(1) but the eigenvalues are different in this case which is quite obvious because we have considered the extra condition which is known as wrap up condition or periodic condition i.e. $\sigma_{n+1} \equiv \sigma_1$ in this Hamiltonian.

3.3 Chapter Summary

In this chapter, we discussed quasicrystals, which are different from normal crystals. We also discussed how we can imagine the system mathematically with the help of the inverse Fibonacci series. We also construct the transverse Ising-like model involving inverse Fibonacci-type nearest neighbour interaction energies between the adjacent qubits. Then we discussed the energy eigenvalue spectrum with the variation of the magnetic field for both open chain and periodic chain systems. In the forthcoming chapter, we are going to discuss the entanglement-related study between the qubits in such a type of spin chain system.

Chapter 4

Entanglement Study in Ising-like Spin Chain systems

4.1 Introduction to Quantum Entanglement

Quantum entanglement is a fundamental phenomenon where two or more particles become strongly correlated such that their quantum state cannot be described independently, even when large distances separate them. Let us explain the concept of quantum entanglement[17] simply. Suppose two boys had interacted for some time (Fig.[4.1]), and later they were separated by a large distance. Now, if I pinch one boy, you may observe that the other boy also feels pain for it (Fig.[4.2]), although you have not done anything to him!! It may surprise you. If it is the case, then we can say that the two boys are entangled because of a certain correlation between them. Similarly, when two states or systems are entangled, then measurement on any one of them affects the second one, without doing anything to the second. This fact was pointed out by the observation of Einstein, Podolsky, and Rosen, which is now known as EPR paradox [18]. Later, this phenomenon was given the name entanglement by Schrödinger.

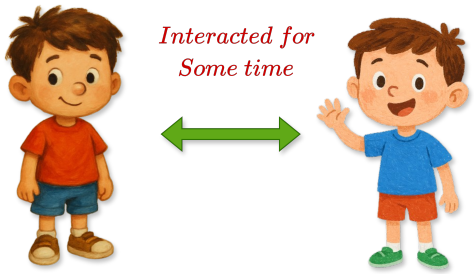


Figure 4.1: Interaction of the two boys for some time.

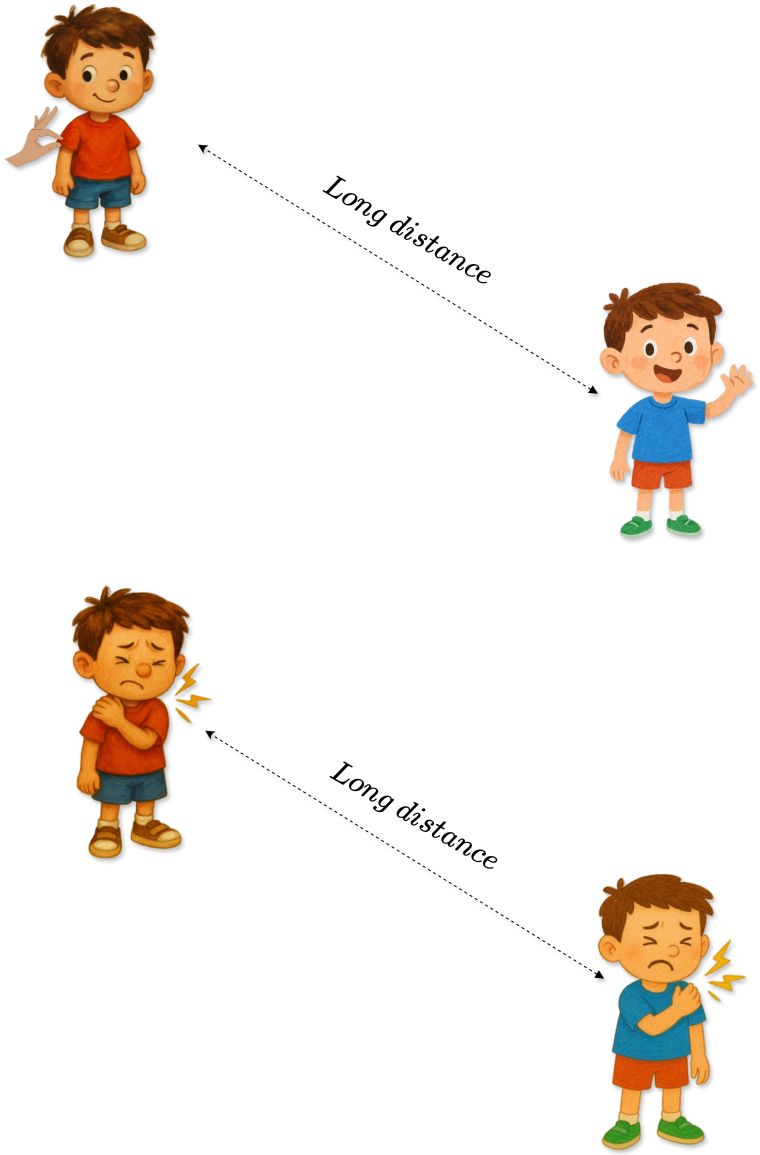


Figure 4.2: Pictorial analogy of entanglement between the two boys .

Mathematically, if H_A and H_B are two finite-dimensional Hilbert spaces, and $|\psi\rangle \in H_A \otimes H_B$, then $|\psi\rangle$ is said to be entangled if there do not exist states $|\psi_A\rangle \in H_A$ and $|\psi_B\rangle \in H_B$ such that

$$|\psi\rangle \neq |\psi_A\rangle \otimes |\psi_B\rangle \tag{4.1}$$

4.1.0.1 Importance of Quantum Entanglement

Quantum entanglement has immense importance nowadays [17]. It finds its application in quantum information theory, quantum cryptography, quantum computing, quantum teleportation (It is a protocol to transmit an unknown quantum state of a qubit using two classical bits such that the receiver reproduces the same state as the original qubit state and it is important to note that qubit itself is not transported rather the information required to produce the quantum state is transmitted Fig.[4.3])etc.

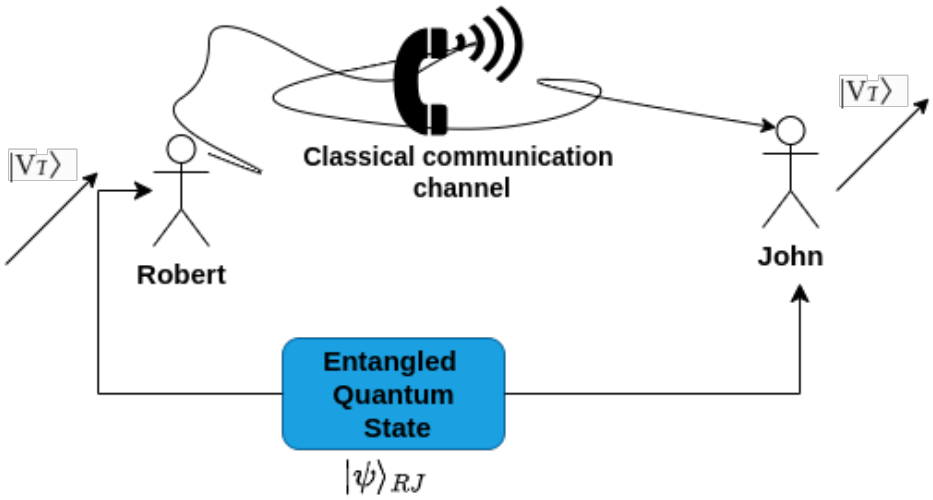


Figure 4.3: Quantum Teleportation

4.2 Entanglement Measure

It becomes sometimes imperative to measure the entanglement, and there are various methods now to measure the entanglement both for bipartite and multipartite

systems. In the present scenario, we are going to discuss the entanglement between two qubits involved in the system that we had considered in Chapters 3 and 4. For that purpose, the effective measurement tools are Concurrence and von Neumann entropy.

4.2.1 Concurrence

Concurrence is a measure of entanglement that was introduced by Wooters(1997) and is defined for both pure and mixed states of two qubits [19]. It is given by,

$$C_{ij} = \text{Max}(0, \sqrt{\lambda_1} - \sqrt{\lambda_2} - \sqrt{\lambda_3} - \sqrt{\lambda_4}) \quad (4.2)$$

Where λ_i's are the eigenvalues of the matrix $\rho_{ij}\tilde{\rho_{ij}}$ in non-increasing order and C_{ij} is the concurrence between i^{th} and j^{th} qubits provided $i > j$, ρ_{ij} is the reduced density matrix corresponding to the qubits i and j between which we are calculating the concurrence and $\tilde{\rho_{ij}}$ is defined as

$$\tilde{\rho_{ij}} = (\sigma_y \otimes \sigma_y)\rho_{ij}^{*}(\sigma_y \otimes \sigma_y) \quad (4.3)$$

σ_y is the Y component of the Pauli spin matrix, which can be given by, $\sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$.

The value of concurrence(C_{ij}) lies between 0 and 1 and

$$C_{ij} = \begin{cases} 0 & \text{for separable state} \\ 1 & \text{for a maximally entangled state} \end{cases}.$$

4.2.2 von Neumann Entropy and Block Entropy

For a two-qubit pure state, we used to consider von Neumann entropy [20][21], also known as entanglement entropy, as an entanglement measure. Suppose we have two qubits in the Hilbert space $H = H_A \otimes H_B$ where, A stands for the first qubit and B stands for the second qubit, and the state of the composite system of two qubits is given by ρ_{AB}, which is a 4×4 density matrix. We use the reduced density matrices

ρ_A and ρ_B and find the von Neumann entropy as follows to evaluate the entanglement content between A and B.

$$S(\rho_A) = S(\rho_B) = -\sum_{i=1}^{2} \eta_i log_2 \eta_i \tag{4.4}$$

. Here, η_i's are the nonzero eigenvalues of ρ_A or ρ_B, which will be equal to the Schmidt decomposition. The total entropy of a maximally entangled composite system is zero, but the entropies for the two subsystems are unity.

For a multiqubit pure state, the concept of Block entropy comes into the picture, which is nothing but the extension of von Neumann entropy only. Suppose we are considering an N qubit pure state, then we can split the system into two subsystems like K and $N-K$, where $K < N$.

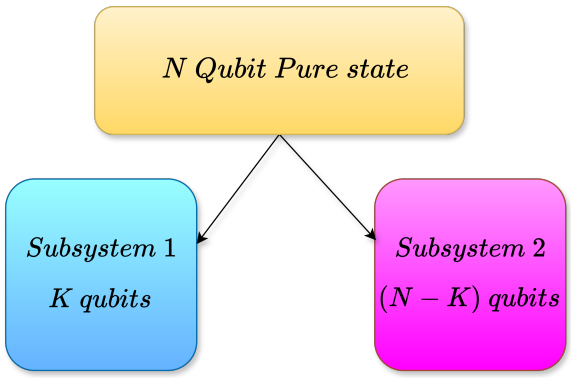


Figure 4.4: Division of the total system into two blocks.

Now we can find the block entropy corresponding to these two blocks as follows,

$$S(\rho_K) = -\mathrm{Tr}(\rho_K \log_2 \rho_K) = -\sum_{i=1}^{\dim(\rho_K)} \eta_i' \log_2 \eta_i' \tag{4.5}$$

$$S(\rho_{N-K}) = -\mathrm{Tr}(\rho_{N-K} \log_2 \rho_{N-K}) = -\sum_{i=1}^{\dim(\rho_{N-K})} \eta_i'' \log_2 \eta_i'' \tag{4.6}$$

lambda Here, $\eta_i' s$ and $\eta_i^{"}$s are the non-zero eigenvalues of the matrices ρ_K or ρ_{N-K} respectively which are equal.

4.2.3 Monogamy Of Entanglement(MOE)

Quantum entanglement can not be shared among many parties like classical correlation because of certain kinds of restrictions. This restricted shareability of entanglement is known as monogamy of entanglement(MOE) [22]. Let us understand this visually. From the Fig.[4.5], it can be seen that when the two parties, which are designated in the green color, are maximally entangled, then the third party, say the red one, can not interfere in their entanglement. Whereas if they don't possess strong entanglement, then the third party can easily interfere.

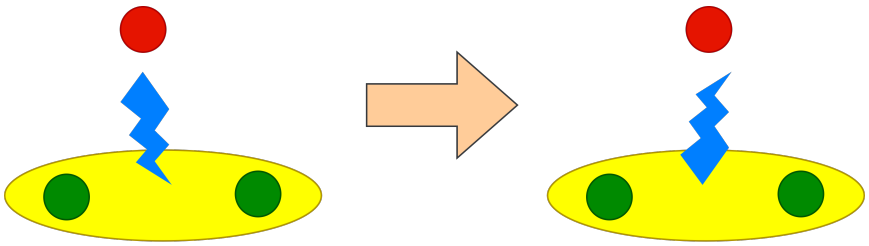


Figure 4.5: Monogamy of Entanglement

4.3 Entanglement Study in Transverse Ising-like model with the Inverse Fibonacci type of Interactions

We have already discussed the Hamiltonian of this kind of system in Chapter 3, where we were mainly focused on the interaction energies taken along the Z-direction and the applied magnetic field taken transverse to it (in the X-direction), and it was given as follows,

$$\mathcal{H} = \sum_i J_i \sigma_i^z \sigma_{i+1}^z + B \sum_i \sigma_i^x \quad . \tag{4.7}$$

You are already familiar with the terms. In the previous chapter, we also considered the two cases of open and periodic chain systems. Now, we are going to discuss the entanglement between a pair of qubits present in such systems.

4.3.1 Entanglement Study for systems consisting of N qubits

To study the concurrence corresponding to all combinations of pairs of qubits numerically, we can use the Fortran code given below. We are interested in studying these concurrences corresponding to the ground state of the system.

4.3.1.1 Case-(1) For Open Chain System

For the open chain configuration, what will be the scenario? Let us see first. We can construct the following Fortran code to study that one.

```
program study_of_concurrence_for_Isinglike_Hamiltonian
    !Program of generalized Isinglike Model when J is varied like
   inverse Fibonacci series and B, is also varied
    !Open chain System
    implicit none
    integer :: n, i, num, a, x, y, k, differing_bits
    integer, parameter :: t=0, c=1
    real*8 :: B,  con, be1, be2
    integer :: m, count
    character(len=:), allocatable :: state(:)
    integer, allocatable :: decimal(:)
    integer, allocatable :: psi(:)
    double precision, allocatable :: p(:), vec1(:), res1(:), vit(:)
    integer :: j
    real*8, allocatable :: S1(:,:), S2(:,:), H(:,:), H1(:,:), S3
   (:,:),rho(:,:), Ham(:,:),lrho(:,:), res(:,:)

    !For PTRVR
    integer :: s
    integer, parameter :: d = 2
    real*8 :: rdm(0:2**d-1, 0:2**d-1)
    integer, allocatable :: site(:)

    ! Parameters for LAPACK DSYEV
    integer :: lda, info, lwork, lda1, lwork1
    real*8, allocatable :: w(:), work(:), w1(:),  work1(:),w2(:),
```

```
    work2(:)
25   character*1 :: JOBZ, UPLO
26
27   ! Input values
28   print*, "Enter the number of bits you want:"
29   read*, n
30   m = n
31   ! Calculate the number of states
32   num = 2**n
33   s = n
34
35   ! Allocate arrays after determining the value of num
36   allocate(character(len=n) :: state(num))
37   allocate(integer :: decimal(num))
38   allocate(psi(n))
39   allocate(S1(num, num))
40   allocate(S2(num, num))
41   allocate(S3(num, num))
42   allocate(H(num, num))
43   allocate(H1(num, num))
44   allocate(p(m))
45   allocate(rho(0:num-1,0:num-1))
46   allocate(lrho(0:num-1,0:num-1))
47   allocate(vec1(num))
48   allocate(vit(0:num-1))
49   allocate(res1(num))
50   allocate(Ham(num,num))
51   allocate(site(0:d-1))
52   allocate(res(0:num-1,0:num-1))
53
54   ! LAPACK parameters
55   lda = num
56   lwork = 3 * num - 1
57   lda1 = num-1
58   lwork1 = 3*(num-1)-1
59   allocate(real(8) :: w(lda), work(lwork))
60   allocate(real(8) :: w1(lda), work1(lwork))
61
```

```
62      ! Printing the possible states
63      !print*, "The states are:"
64      do i = 0, num - 1
65          call integer_binary(i, state(i + 1), n)  ! Adjusting index
    for 1-based
66      end do
67
68      ! Convert binary states to decimal
69      do i = 1, num
70          decimal(i) = btod(state(i), n)
71      end do
72
73      call FibonacciSeries(m, p)
74      ! Perform calculations based on the decimal values for S1
75       do i = 1, num
76          do j = 1, num
77              call single_digit(state(i), psi)
78              if (decimal(i) /= decimal(j)) then
79                  S1(i, j) = 0.0
80              else
81                  S1(i, j) = 0.0
82                  do k = 2, n
83                      S1(i, j) = S1(i, j) + p(k-1)*(1.0 - 2.0 * psi(k
    -1)) * (1.0 - 2.0 * psi(k))
84                  end do
85              end if
86          end do
87      end do
88
89      ! Perform calculations based on the decimal values for S2
90      if (t /= 0) then
91          do i = 1, num
92              do j = 1, num
93                  call single_digit(state(i), psi)
94                  if (decimal(i) /= decimal(j)) then
95                      S2(i, j) = 0.0
96                  else
97                      S2(i, j) = 0.0
```

```
                    do k = 1, n
                        S2(i, j) = S2(i,j) + (1.0 - 2.0 * psi(k))
                    end do
                end if
            end do
        end do
    end if

    ! Construct the S3 matrix (for sigma_x )
    S3 = 0.0
    do i = 1, num
        do j = 1, num
            ! Initialize differing_bits counter
            differing_bits = 0
            do k = 1, n
                ! Compare the kth bit of the i-th and j-th states
                if (btest(decimal(i), k-1) .neqv. btest(decimal(j),
k-1)) then
                    differing_bits = differing_bits + 1
                end if
            end do
            ! If exactly one bit differs, set S2(i, j) = 1.0
            if (differing_bits == 1) then
                S3(i, j) = 1.0
            else
                S3(i, j) = 0.0
            end if
        end do
    end do

    ! Total Hamiltonian
    do y = 0, 100, 1
        B = y * 0.01
        H =  S1 + B * (t * S2 + c * S3)
        do i = 1, num
            do j = 1, num
                Ham(i,j) = H(i,j)
            end do
```

```
135         end do
136
137         ! Eigenvalue and eigenvector determination using DSYEV
138         JOBZ = 'V'
139         UPLO = 'U'
140         call DSYEV(JOBZ, UPLO, num, H, lda, w, work, lwork, info)
141         vec1 = H(:, 1)
142         do i = 1, num
143             vit(i-1) = vec1(i)
144         end do
145
146         !Concurrence calculation
147         do i = 0, n-1
148             do j = i+1, n-1
149                 if( i /= num) then
150                     call CONVR(n,i,j,vit,con)
151                     !For Magnetic field vs concurrence plot
152                     write(40,*) B, con
153                 else
154                     EXIT
155                 end if
156             end do
157         end do
158     end do
159
160     deallocate(state)
161     deallocate(decimal)
162     deallocate(psi)
163     deallocate(S1)
164     deallocate(S2)
165     deallocate(S3)
166     deallocate(H)
167     deallocate(H1)
168     deallocate(p)
169     deallocate(rho)
170     deallocate(lrho)
171     deallocate(vec1)
172     deallocate(site)
```

```
173     deallocate(vit)
174 contains
175     function btod(state, n) result(decimal)
176         implicit none
177         integer, intent(in) :: n
178         character(len=n) :: state
179         integer :: decimal, i
180         decimal = 0
181         do i = 1, n
182             if (state(i:i) == '1') then
183                 decimal = decimal + 2**(n-i)
184             end if
185         end do
186     end function btod
187 
188     subroutine integer_binary(num, binary_string, n)
189         implicit none
190         integer :: num
191         integer, intent(in) :: n
192         integer :: i, temp_num
193         character(len=n), intent(out) :: binary_string
194         temp_num = num
195         binary_string = repeat('0', n)
196         do i = 0, n - 1
197             if (mod(temp_num, 2) == 1) then
198                 binary_string(n-i:n-i) = '1'
199             else
200                 binary_string(n-i:n-i) = '0'
201             end if
202             temp_num = temp_num / 2
203         end do
204     end subroutine integer_binary
205 
206     subroutine single_digit(A, psi)
207         implicit none
208         character(len=*), intent(in) :: A
209         integer, intent(out) :: psi(:)
210         integer :: i
```

```
211         do i = 1, len(A)
212             if (A(i:i) == '0') then
213                 psi(i) = 0
214             else if (A(i:i) == '1') then
215                 psi(i) = 1
216             else
217                 print *, "Invalid input in single_digit subroutine"
218             end if
219         end do
220     end subroutine single_digit
221 
222     subroutine FibonacciSeries(m, reciprocal)
223         implicit none
224         integer :: i
225         integer :: m
226         double precision :: fib(m)
227         double precision :: reciprocal(m)
228         if (m <= 0) then
229             print*, "Number of terms must be greater than 0."
230             stop
231         end if
232         ! Initialize the first two terms
233         fib(1) = 1
234         if (n > 1) then
235             fib(2) = 1
236         end if
237         ! Generate the Fibonacci series
238         do i = 3, m
239             fib(i) = fib(i-1) + fib(i-2)
240         end do
241         ! Compute reciprocals
242         do i = 1, m
243             reciprocal(i) = 1.0d0 / real(fib(i))
244         end do
245         !print*, fib
246     end subroutine FibonacciSeries
247 
248     subroutine MATRIXMUL(A, B, m, n, p, C)
```

```
249         implicit none
250         integer :: m, n, p, i, j, k
251         real*8 :: A(m,n), B(n,p), C(m,p), temp
252 
253         do i = 1, m, 1
254             do j = 1, p, 1
255                 temp = 0.0d0
256                 do k = 1, n, 1
257                     temp = temp + A(i,k)*B(k,j)
258                 end do
259                 C(i,j) = temp
260             end do
261         end do
262         !write(1237,*) C
263         end subroutine
264 
265         subroutine PTRVR(s, s1, site, vin, rdm)
266             implicit none
267             integer :: s, s1, s2
268             real*8 :: trace
269             real*8, dimension(0:2**s-1) :: vin
270             real*8, dimension(0:2**s1-1, 0:2**s1-1) :: rdm
271             integer :: ii, i1, a, i0, ia, j1, oo, k1, x1, y1, j, t1,
    t2, z1
272             integer, dimension(0:s1-1) :: site, site2
273             integer, dimension(0:s1-1) :: bin
274             integer, dimension(0:(2**(s-s1))*(2**s1)-1) :: ind
275             s2 = s - s1
276             x1 = 0
277             y1 = 0
278             ind = 0
279 
280             do j1 = 2**s1-1, 0, -1
281                 do ii = 0, 2**s-1
282                     a = ii
283                     do i1 = 0, s1-1
284                         i0 = site(i1)
285                         call DTOBONEBIT(a, ia, i0, s)
```

```
286                         site2(i1) = ia
287                     end do
288                     call DTOB(j1, bin, s1)
289                     oo = 1
290                     do k1 = 0, s1-1
291                         oo = oo * (bin(k1) - site2(k1))
292                     end do
293                     if (abs(oo) == 1) then
294                         ind(x1) = ii
295                         x1 = x1 + 1
296                     end if
297                 end do
298             end do
299
300             rdm = 0.0d0
301             do t1 = 0, 2**s1-1
302                 do t2 = 0, 2**s1-1
303                     do z1 = 0, 2**s2-1
304                         rdm(t1, t2) = rdm(t1, t2) + vin(ind((2**s2)
    * t1 + z1)) * vin(ind((2**s2) * t2 + z1))
305                     end do
306                 end do
307             end do
308         end subroutine
309
310         subroutine DTOBONEBIT(m, ia, i0, s)
311             implicit none
312             integer*4 :: s
313             integer :: tt(0:s-1)
314             integer :: m, ia, i0
315             call DTOB(m, tt, s)
316             ia = tt(i0)
317         end subroutine
318
319         subroutine DTOB(m, tt, s)
320             implicit none
321             integer :: s
322             integer, dimension(0:s-1) :: tt
```

```
323             integer :: m, k, a2
324             tt = 0
325             a2 = m
326             do k = 0, s-1
327                 tt(s-k-1) = mod(a2, 2)
328                 a2 = a2 / 2
329                 if (a2 == 0) exit
330             end do
331         end subroutine
332
333         subroutine CONVR(s,i0,j0,vin,con)
334             implicit none
335             integer::s,i0,j0 ,site(0:1),i
336             real*8::vin(0:2**s-1),evs
337             real*8,dimension(0:3,0:3)::rdm,MM1,MM2,MM3,MM4
338             INTEGER::INFO, LDA, LWORK, NN,LDVL,LDVR
339             parameter(NN=4,LDA=NN,LWORK=4*NN,LDVL=NN,LDVR=NN)
340             real*8::WORK(0:LWORK-1),AA2(0:LDA-1,0:NN-1)
341             real*8::VL(0:NN-1,0:NN-1),VR(0:NN-1,0:NN-1)
342             real*8::WR(0:NN-1),WI(0:NN-1)
343             real*8::ev(NN) ,con,c2,x11,const,sigmay(0:3,0:3)
344             sigmay(0,:)=(/0.0,0.0,0.0,-1.0/)
345             sigmay(1,:)=(/0.0,0.0,1.0,0.0/)
346             sigmay(2,:)=(/0.0,1.0,0.0,0.0/)
347             sigmay(3,:)=(/-1.0,0.0,0.0,0.0/)
348             site(0)=i0
349             site(1)=j0
350             call PTRVR(s,2,site,vin,rdm)
351             MM1=matmul(rdm,sigmay)
352             MM2=matmul(rdm,sigmay)
353             MM3=matmul(MM1,MM2)
354             call dgeev('N', 'N', NN, MM3, LDA, WR, &
355             WI, VL, LDVL, VR, LDVR, WORK, LWORK, INFO)
356             do i=1,NN,1
357                 if (WR(i-1) .lt. 0.0000000000001) then
358                     WR(i-1)=0.0d0
359                 end if
360                 ev(i)=WR(i-1)
```

```
361             end do
362             do i=2,4
363                 if(ev(i)>=ev(1))then
364                     evs=ev(i)
365                     ev(i)=ev(1)
366                     ev(1)=evs
367                 end if
368             end do
369             do i=3,4
370                 if(ev(i)>=ev(2))then
371                     evs=ev(i)
372                     ev(i)=ev(2)
373                     ev(2)=evs
374                 end if
375             end do
376             do i=4,4
377                 if(ev(i)>=ev(3))then
378                     evs=ev(i)
379                     ev(i)=ev(3)
380                     ev(3)=evs
381                 end if
382             end do
383             c2=0
384             con=0
385             c2=dsqrt(ev(1))-dsqrt(ev(2))-dsqrt(ev(3))-dsqrt(ev(4))
386             if(c2>0)then
387                 con=c2
388             else
389                 con=0
390             end if
391         end subroutine
392 end program program study_of_concurrence_for_Isinglike_Hamiltonian
```

Let us first consider a three-qubit open chain system (by choosing n of the above program to be 3). We have calculated the concurrences between the qubit pairs the 1st and 2nd qubits (C_{12}), the 2nd and 3rd qubits (C_{23}), and the 1st and 3rd qubits (C_{13}). To study the concurrences, let us visualize them through the plot given below.

We can also see their variations with the change in the applied magnetic field in Fig.[4.6].

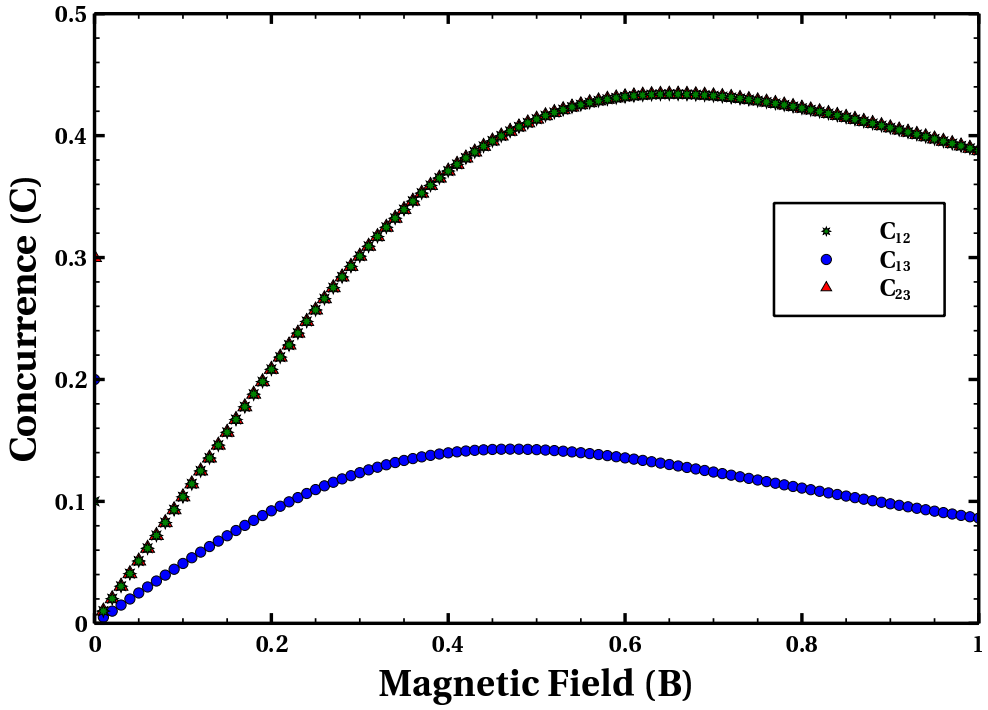
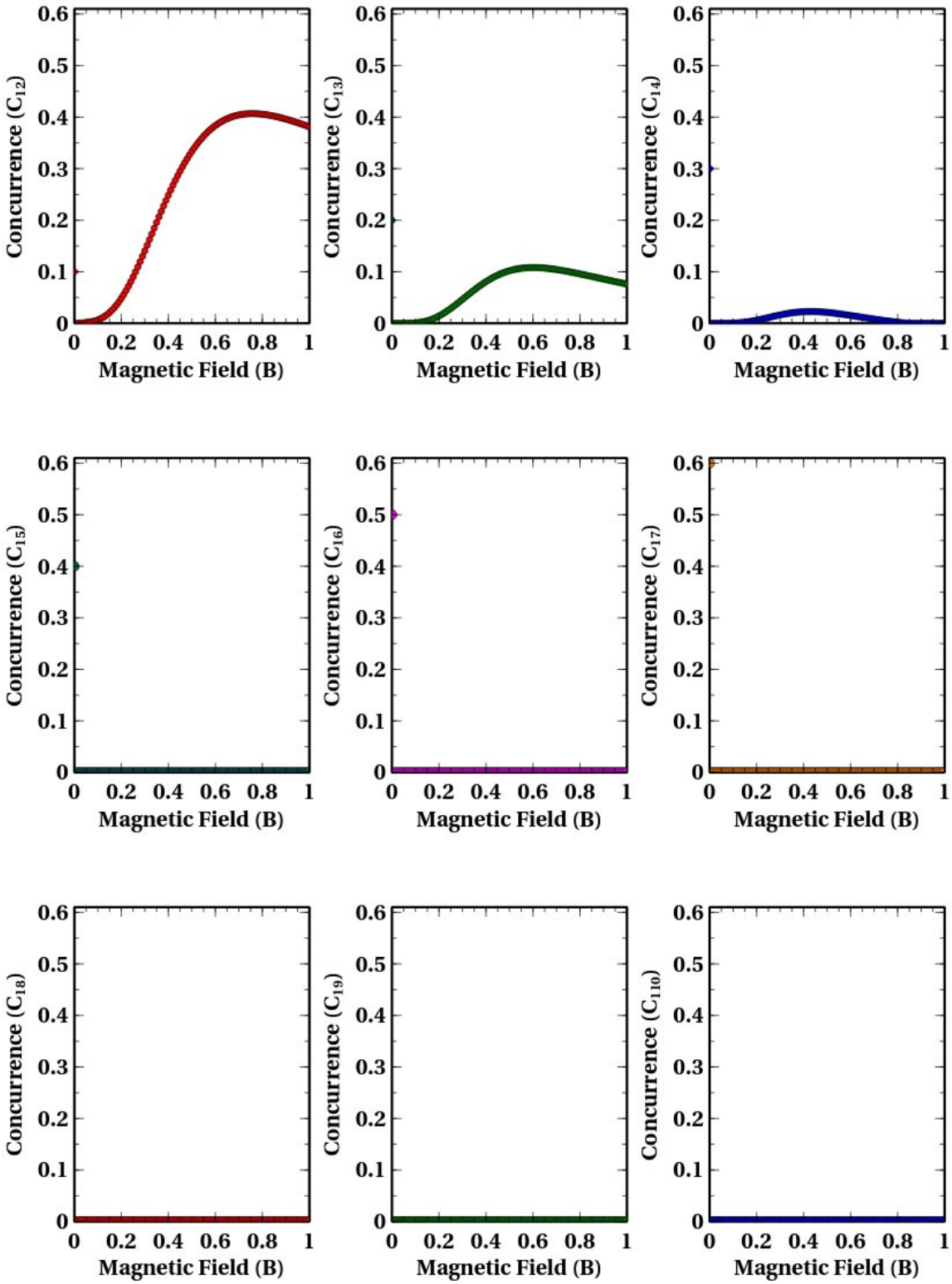
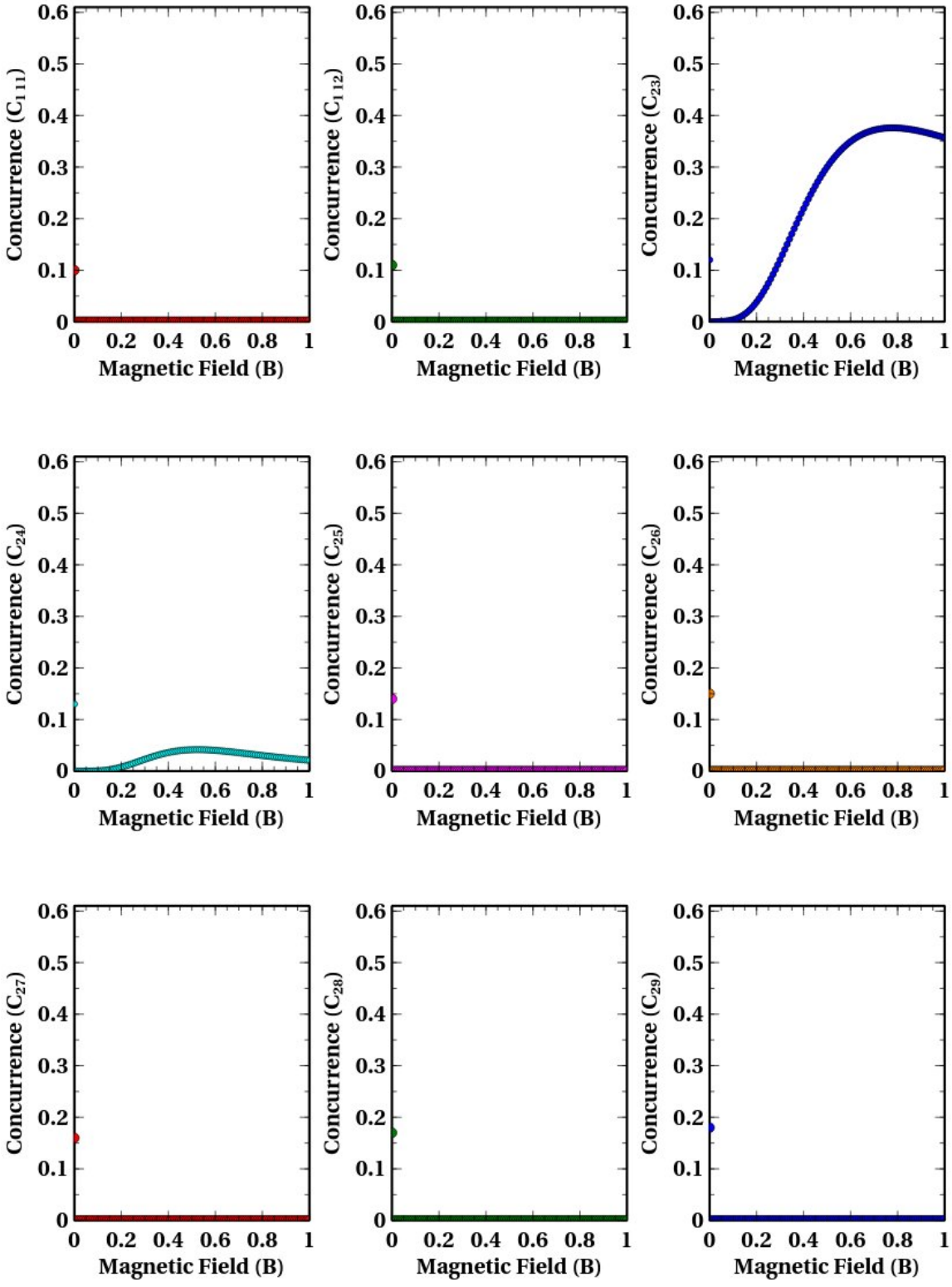


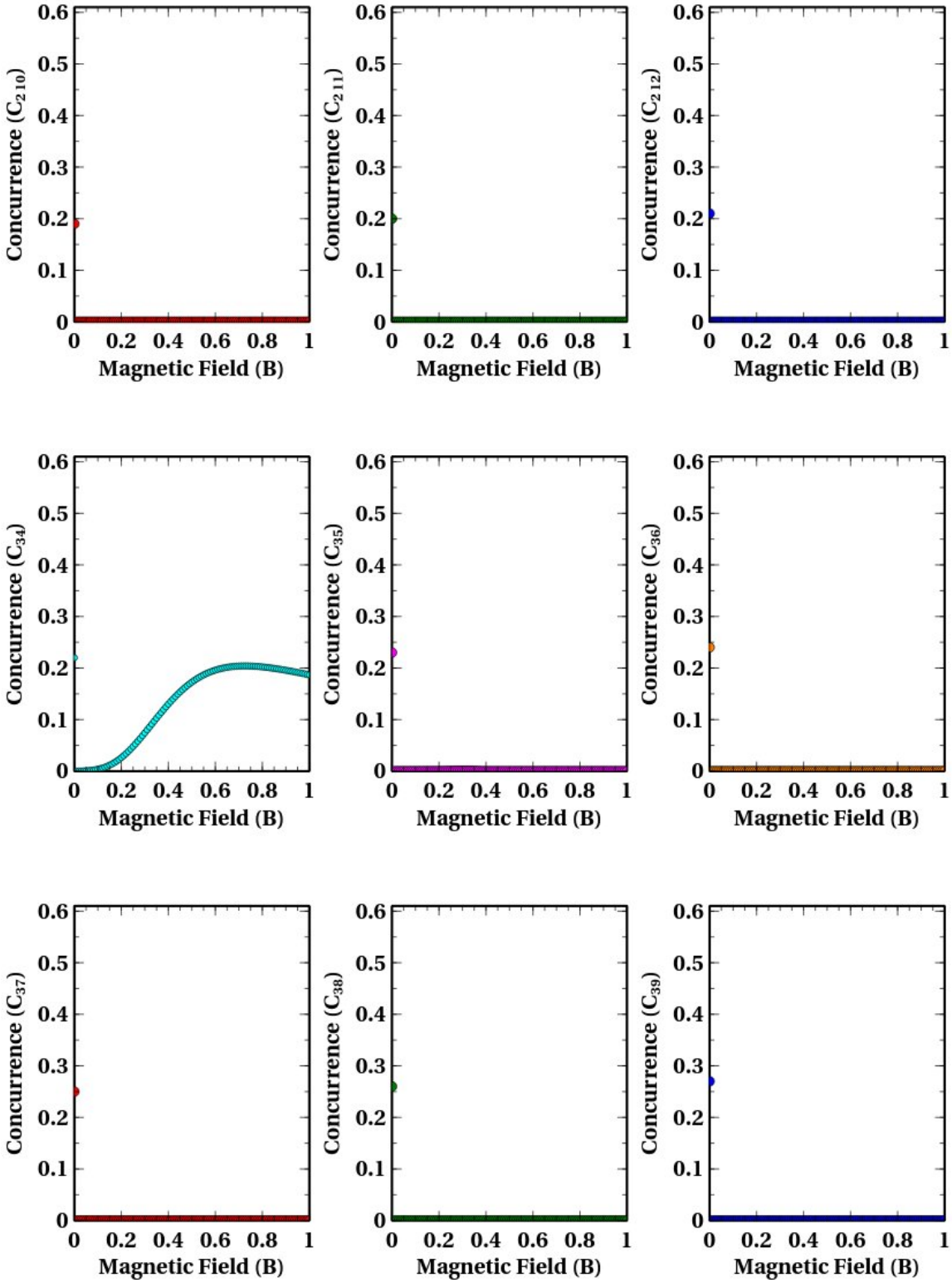
Figure 4.6: Variation of Concurrences for the 3-qubit open chain Ising-like System

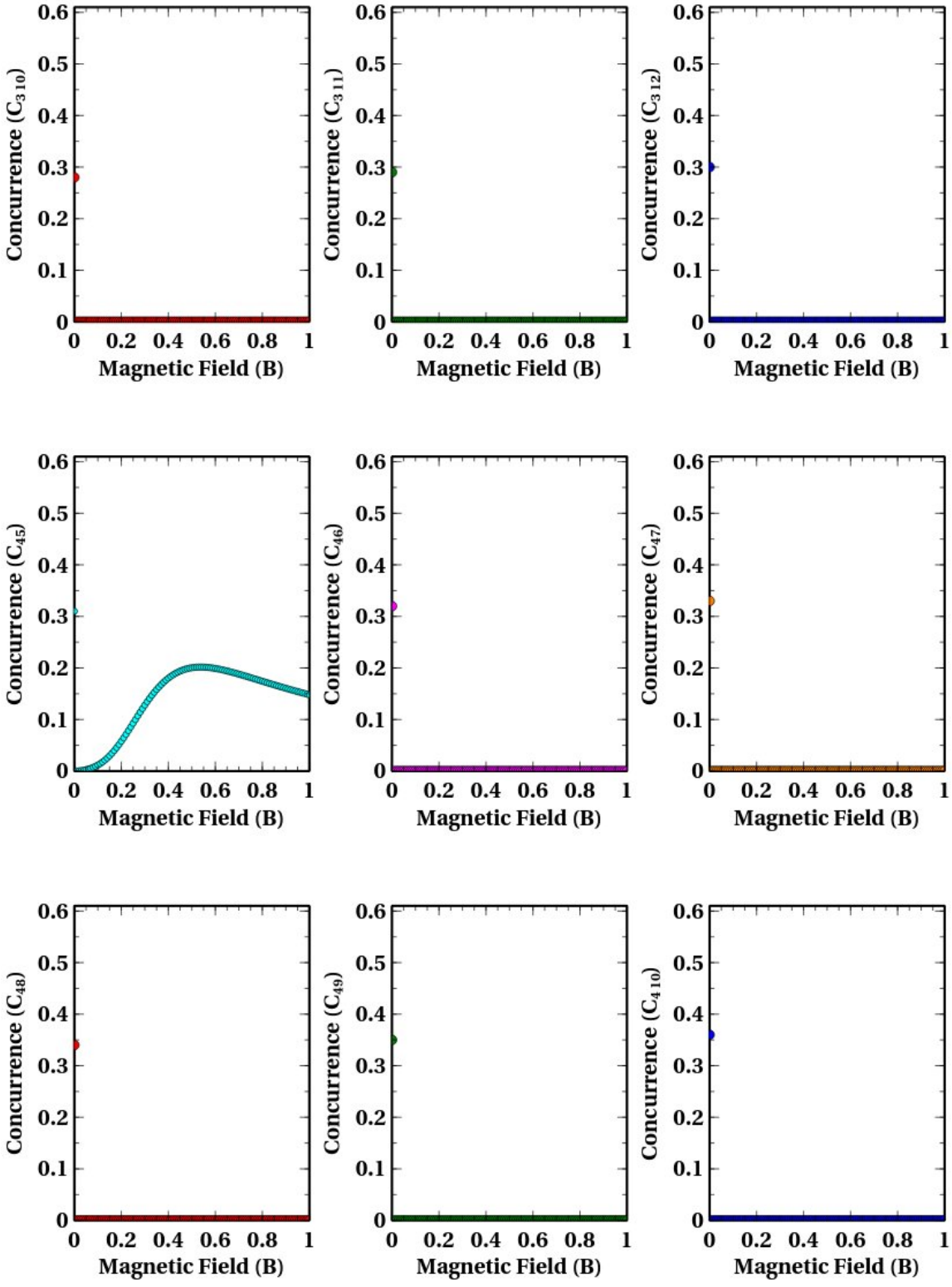
We can see from Fig.[4.6] that the concurrences between the 1st and 2nd qubit (C_{12}), 2nd and 3rd qubit (C_{23}) increases in the same way with the change of magnetic field which signifies the degree of entanglement is also increases in the same way whereas the concurrences between the 1st and 3rd qubit (C_{13}) also increases with magnetic field but the rate is very small compared to C_{12} and C_{23}. We can also go for the systems consisting higher number of qubits; for example, we can consider the 12-qubit system and show the variation of concurrences between different pairs of qubits as follows.

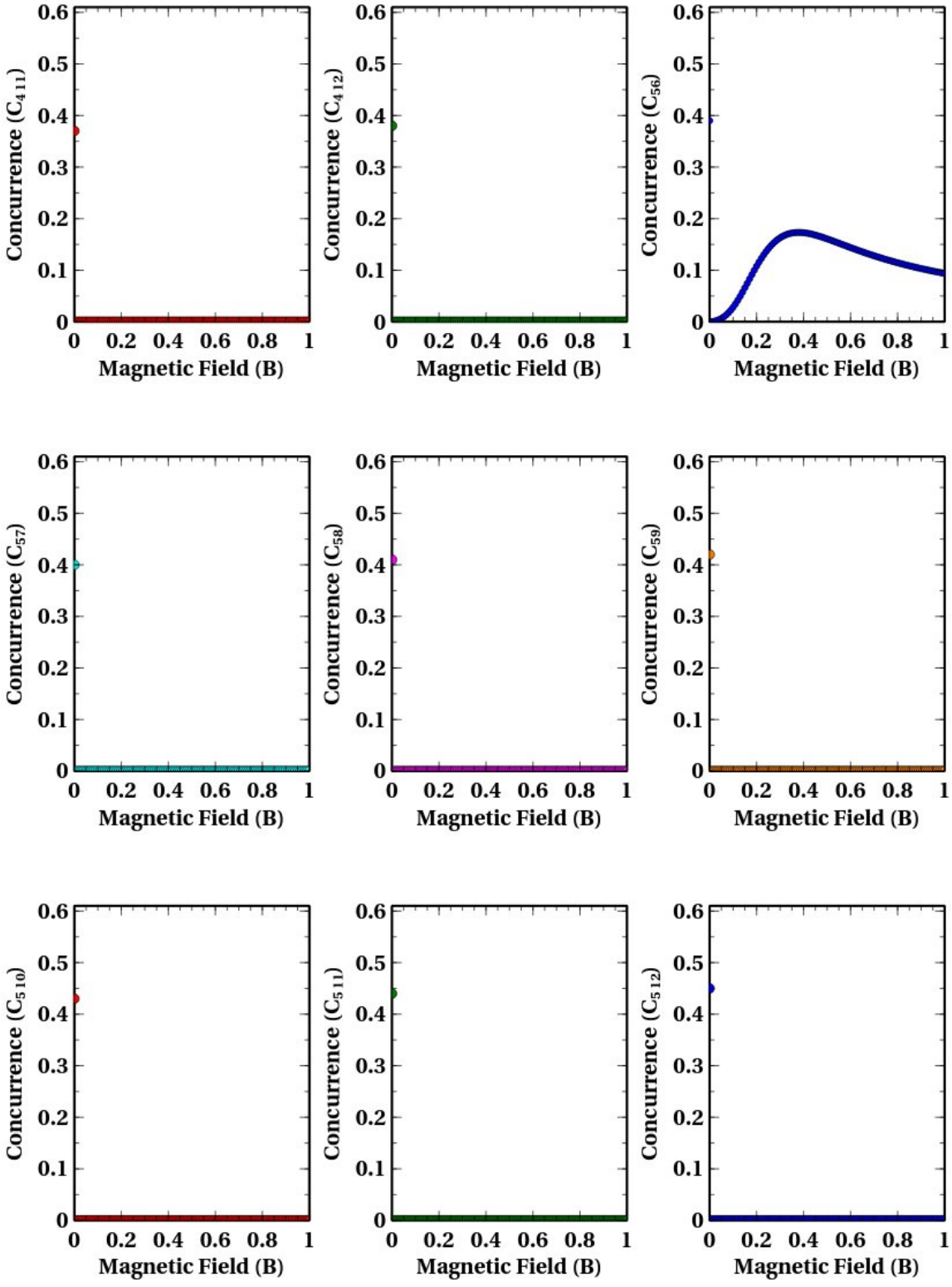
4.3.1.2 Concurrence study for 12 qubit open chain Ising-like system

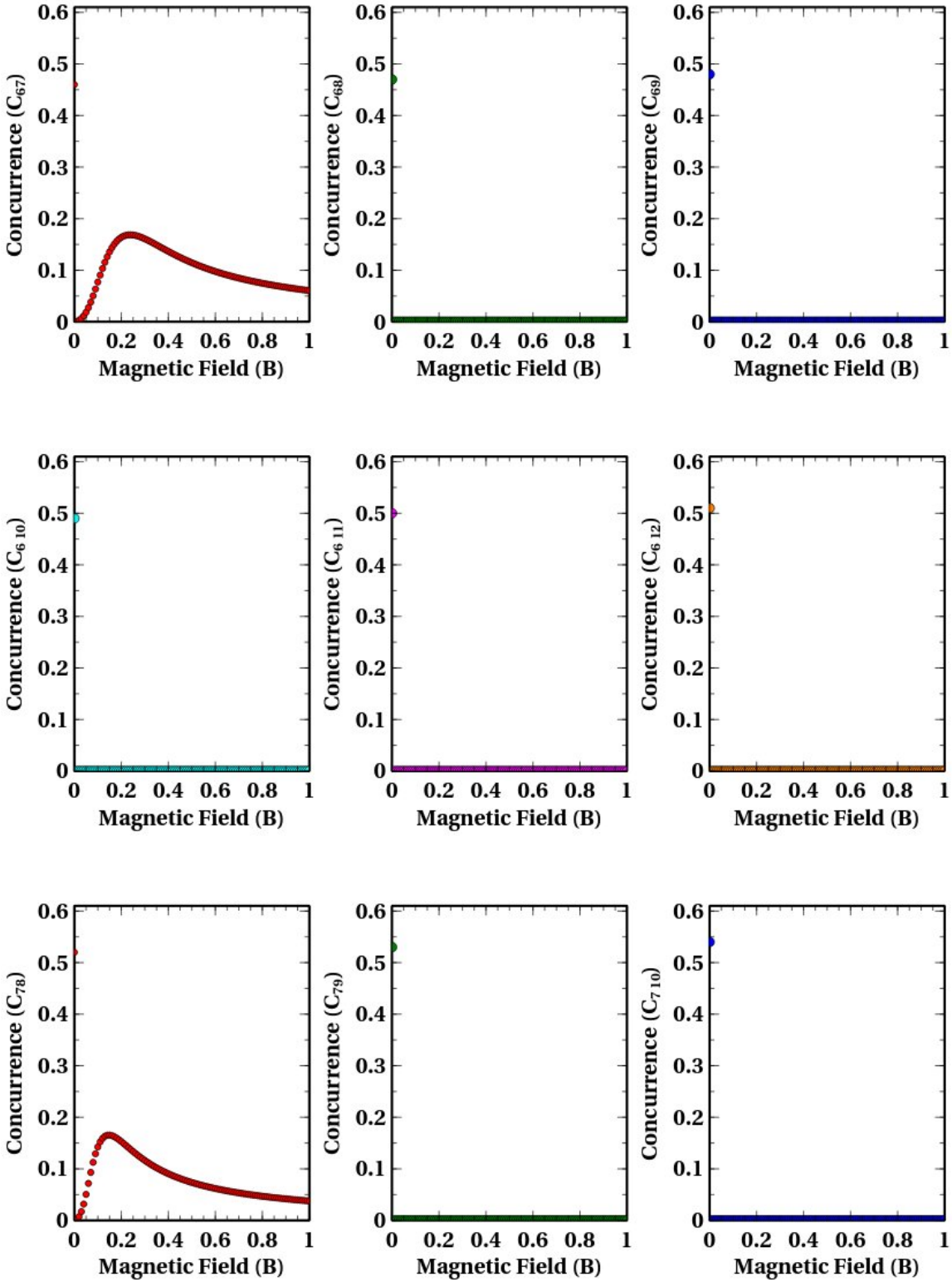


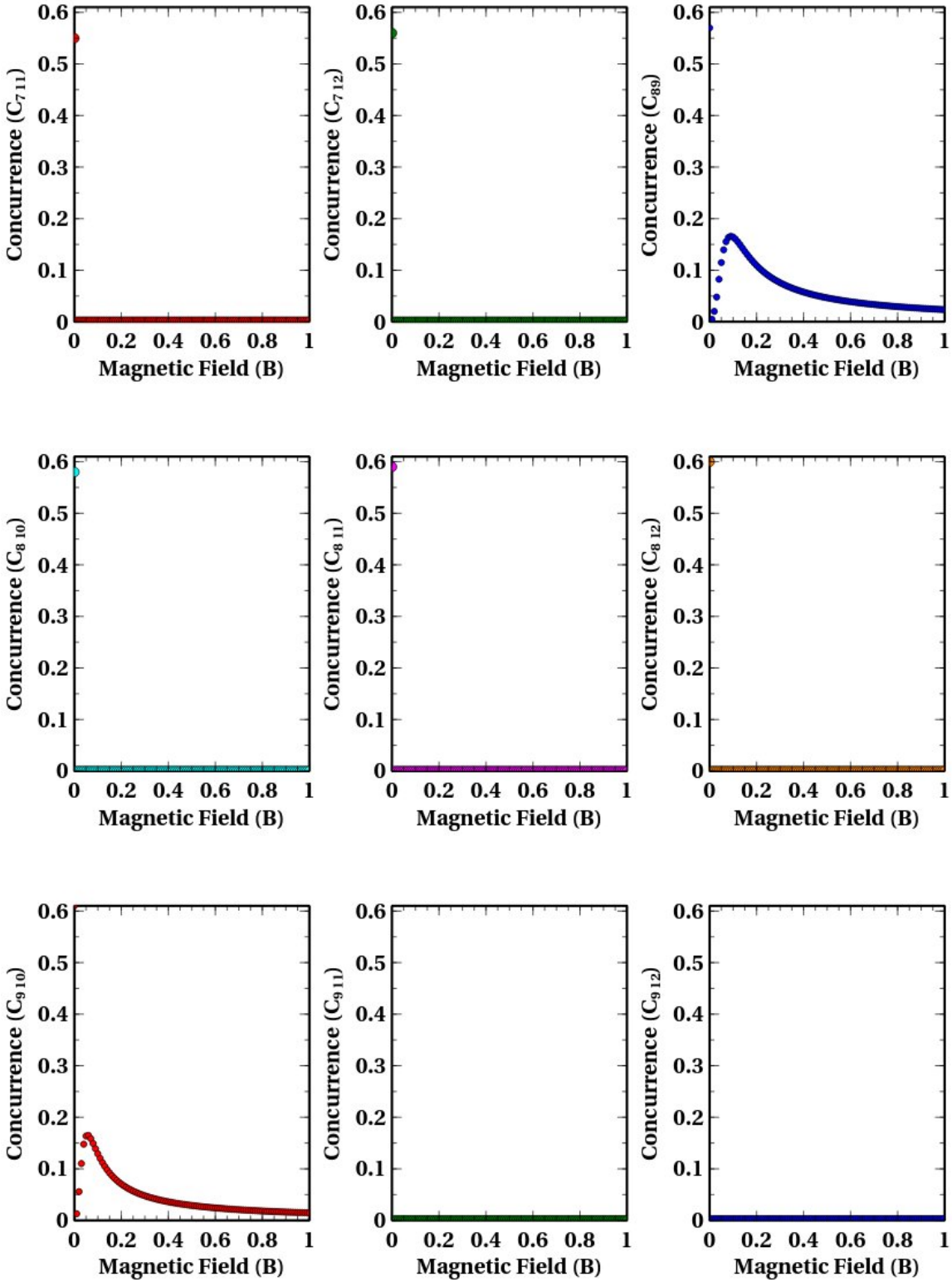

Concurrence ($C_{1\,11}$)
Concurrence ($C_{1\,12}$)
Concurrence (C_{23})
Concurrence (C_{24})
Concurrence (C_{25})
Concurrence (C_{26})
Concurrence (C_{27})
Concurrence (C_{28})
Concurrence (C_{29})
Magnetic Field (B)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.8
1

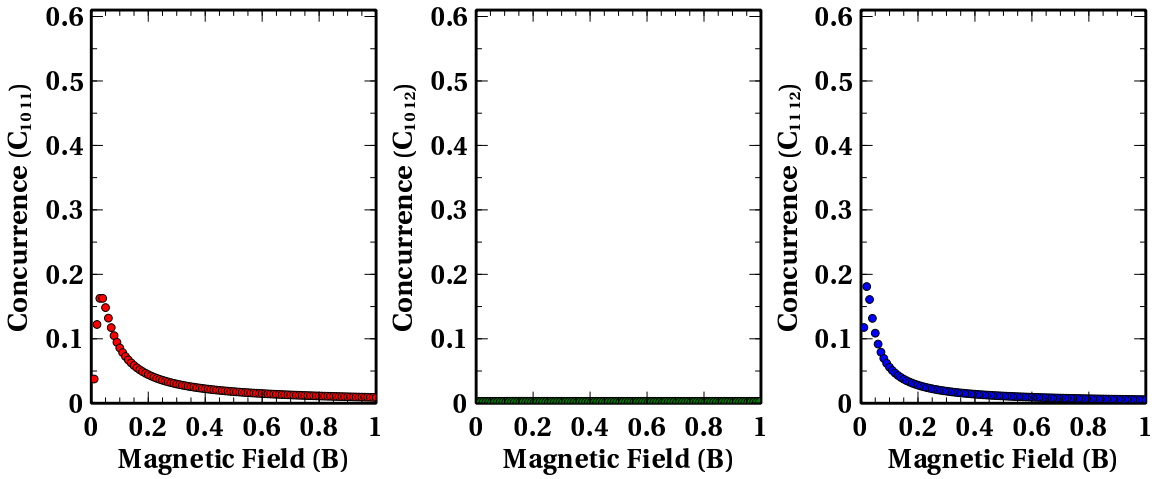

Concurrence (C_{210})
Concurrence (C_{211})
Concurrence (C_{212})
Concurrence (C_{34})
Concurrence (C_{35})
Concurrence (C_{36})
Concurrence (C_{37})
Concurrence (C_{38})
Concurrence (C_{39})
Magnetic Field (B)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.2
0.4
0.6
0.8
1


Concurrence (C_{310})
Concurrence (C_{311})
Concurrence (C_{312})
Concurrence (C_{45})
Concurrence (C_{46})
Concurrence (C_{47})
Concurrence (C_{48})
Concurrence (C_{49})
Concurrence (C_{410})
Magnetic Field (B)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.2
0.4
0.6
0.8
1


Concurrence (C_{411})
Concurrence (C_{412})
Concurrence (C_{56})
Concurrence (C_{57})
Concurrence (C_{58})
Concurrence (C_{59})
Concurrence (C_{510})
Concurrence (C_{511})
Concurrence (C_{512})
Magnetic Field (B)
0 0.1 0.2 0.3 0.4 0.5 0.6
0 0.2 0.4 0.6 0.8 1


Concurrence (C_{67})
Concurrence (C_{68})
Concurrence (C_{69})
Concurrence ($C_{6\,10}$)
Concurrence ($C_{6\,11}$)
Concurrence ($C_{6\,12}$)
Concurrence (C_{78})
Concurrence (C_{79})
Concurrence ($C_{7\,10}$)
Magnetic Field (B)
0 0.1 0.2 0.3 0.4 0.5 0.6
0 0.2 0.4 0.6 0.8 1


Concurrence ($C_{7\,11}$)
Concurrence ($C_{7\,12}$)
Concurrence (C_{89})
Concurrence ($C_{8\,10}$)
Concurrence ($C_{8\,11}$)
Concurrence ($C_{8\,12}$)
Concurrence ($C_{9\,10}$)
Concurrence ($C_{9\,11}$)
Concurrence ($C_{9\,12}$)
Magnetic Field (B)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.2
0.4
0.6
0.8
1



In this case, we are also observing a similar increase in concurrences with the rise in a magnetic field, mostly for the nearest neighbour qubit pairs, which have the defined interactions between them as provided by us. Whereas for distant qubit pairs, we are getting zero concurrences, also they are not affected by the external magnetic field. From the above plots, it becomes clear that the entanglement between the nearest neighbour qubit pair is more than that formed between adjacent qubits, and after a certain amount of magnetic field, the entanglement between the distant qubit pairs goes down.

4.3.1.3 Case-(2) For Periodic Chain System

For the periodic chain system, we can implement the wrap-up or periodic condition $\sigma_{n+1} = \sigma_1$ to the previous code and modify as follows.

```
program study_of_concurrence_for_Isinglike_Hamiltonian
    !Program of generalized Isinglike Model, when J is varied like
   inverse Fibonacci series and B, is also varied
    !Periodic chain System
    implicit none
    integer :: n, i, num, a, x, y, k, differing_bits
    integer, parameter :: t=0, c=1
    real*8 :: B,  con, be1, be2
    integer :: m, count
```

```
    character(len=:), allocatable :: state(:)
    integer, allocatable :: decimal(:)
    integer, allocatable :: psi(:)
    double precision, allocatable :: p(:), vec1(:), res1(:), vit(:)
    integer :: j
    real*8, allocatable :: S1(:,:), S2(:,:), H(:,:), H1(:,:), S3
   (:,:),rho(:,:), Ham(:,:),lrho(:,:), res(:,:)

    !For PTRVR
    integer :: s
    integer, parameter :: d = 2
    real*8 :: rdm(0:2**d-1, 0:2**d-1)
    integer, allocatable :: site(:)

    ! Parameters for LAPACK DSYEV
    integer :: lda, info, lwork, lda1, lwork1
    real*8, allocatable :: w(:), work(:), w1(:),  work1(:),w2(:),
   work2(:)
    character*1 :: JOBZ, UPLO

    ! Input values
    print*, "Enter the number of bits you want:"
    read*, n
    m = n
    ! Calculate the number of states
    num = 2**n
    s = n

    ! Allocate arrays after determining the value of num
    allocate(character(len=n) :: state(num))
    allocate(integer :: decimal(num))
    allocate(psi(n))
    allocate(S1(num, num))
    allocate(S2(num, num))
    allocate(S3(num, num))
    allocate(H(num, num))
    allocate(H1(num, num))
    allocate(p(m))
```

```
45      allocate(rho(0:num-1,0:num-1))
46      allocate(lrho(0:num-1,0:num-1))
47      allocate(vec1(num))
48      allocate(vit(0:num-1))
49      allocate(res1(num))
50      allocate(Ham(num,num))
51      allocate(site(0:d-1))
52      allocate(res(0:num-1,0:num-1))
53
54      ! LAPACK parameters
55      lda = num
56      lwork = 3 * num - 1
57      lda1 = num-1
58      lwork1 = 3*(num-1)-1
59      allocate(real(8) :: w(lda), work(lwork))
60      allocate(real(8) :: w1(lda), work1(lwork))
61
62      ! Printing the possible states
63      !print*, "The states are:"
64      do i = 0, num - 1
65          call integer_binary(i, state(i + 1), n)  ! Adjusting index
    for 1-based
66      end do
67
68      ! Convert binary states to decimal
69      do i = 1, num
70          decimal(i) = btod(state(i), n)
71      end do
72
73      call FibonacciSeries(m, p)
74      ! Perform calculations based on the decimal values for S1
75      !Utilising the periodic condition
76       do i = 1, num
77          do j = 1, num
78              if (decimal(i) /= decimal(j)) then
79                  S1(i, j) = 0.0
80              else
81                  call single_digit(state(i), psi)
```

```
                S1(i, j) = 0.0
                do k = 1, n - 1
                    S1(i, j) = S1(i, j) + p(k)*(1.0 - 2.0 * psi(k))
* (1.0 - 2.0 * psi(k + 1))
                end do
                S1(i, j) = S1(i, j) + p(n)*(1.0 - 2.0 * psi(n)) *
(1.0 - 2.0 * psi(1))
            end if
        end do
    end do


    ! Perform calculations based on the decimal values for S2
    if (t /= 0) then
        do i = 1, num
            do j = 1, num
                call single_digit(state(i), psi)
                if (decimal(i) /= decimal(j)) then
                    S2(i, j) = 0.0
                else
                    S2(i, j) = 0.0
                    do k = 1, n
                        S2(i, j) = S2(i,j) + (1.0 - 2.0 * psi(k))
                    end do
                end if
            end do
        end do
    end if

    ! Construct the S3 matrix (for sigma_x )
    S3 = 0.0
    do i = 1, num
        do j = 1, num
            ! Initialize differing_bits counter
            differing_bits = 0
            do k = 1, n
                ! Compare the k-th bit of the i-th and j-th states
                if (btest(decimal(i), k-1) .neqv. btest(decimal(j),
```

```
    k-1)) then
                    differing_bits = differing_bits + 1
                end if
            end do
            ! If exactly one bit differs, set S2(i, j) = 1.0
            if (differing_bits == 1) then
                S3(i, j) = 1.0
            else
                S3(i, j) = 0.0
            end if
        end do
    end do

    ! Total Hamiltonian
    do y = 0, 100, 1
        B = y * 0.01
        H =  S1 + B * (t * S2 + c * S3)
        do i = 1, num
            do j = 1, num
                Ham(i,j) = H(i,j)
            end do
        end do

        ! Eigenvalue and eigenvector determination using DSYEV
        JOBZ = 'V'
        UPLO = 'U'
        call DSYEV(JOBZ, UPLO, num, H, lda, w, work, lwork, info)
        vec1 = H(:, 1)
        do i = 1, num
            vit(i-1) = vec1(i)
        end do

        !Concurrence calculation
        do i = 0, n-1
            do j = i+1, n-1
                if( i /= num) then
                    call CONVR(n,i,j,vit,con)
                    !For Magnetic field vs concurrence plot
```

```
                    write(41,*) B, con
                else
                    EXIT
                end if
            end do
        end do
    end do

    deallocate(state)
    deallocate(decimal)
    deallocate(psi)
    deallocate(S1)
    deallocate(S2)
    deallocate(S3)
    deallocate(H)
    deallocate(H1)
    deallocate(p)
    deallocate(rho)
    deallocate(lrho)
    deallocate(vec1)
    deallocate(site)
    deallocate(vit)
contains
    function btod(state, n) result(decimal)
        implicit none
        integer, intent(in) :: n
        character(len=n) :: state
        integer :: decimal, i
        decimal = 0
        do i = 1, n
            if (state(i:i) == '1') then
                decimal = decimal + 2**(n-i)
            end if
        end do
    end function btod

    subroutine integer_binary(num, binary_string, n)
        implicit none
```

```
193         integer :: num
194         integer, intent(in) :: n
195         integer :: i, temp_num
196         character(len=n), intent(out) :: binary_string
197         temp_num = num
198         binary_string = repeat('0', n)
199         do i = 0, n - 1
200             if (mod(temp_num, 2) == 1) then
201                 binary_string(n-i:n-i) = '1'
202             else
203                 binary_string(n-i:n-i) = '0'
204             end if
205             temp_num = temp_num / 2
206         end do
207     end subroutine integer_binary
208
209     subroutine single_digit(A, psi)
210         implicit none
211         character(len=*), intent(in) :: A
212         integer, intent(out) :: psi(:)
213         integer :: i
214         do i = 1, len(A)
215             if (A(i:i) == '0') then
216                 psi(i) = 0
217             else if (A(i:i) == '1') then
218                 psi(i) = 1
219             else
220                 print *, "Invalid input in single_digit subroutine"
221             end if
222         end do
223     end subroutine single_digit
224
225     subroutine FibonacciSeries(m, reciprocal)
226         implicit none
227         integer :: i
228         integer :: m
229         double precision :: fib(m)
230         double precision :: reciprocal(m)
```

```
231         if (m <= 0) then
232             print*, "Number of terms must be greater than 0."
233             stop
234         end if
235         ! Initialize the first two terms
236         fib(1) = 1
237         if (n > 1) then
238             fib(2) = 1
239         end if
240         ! Generate the Fibonacci series
241         do i = 3, m
242             fib(i) = fib(i-1) + fib(i-2)
243         end do
244         ! Compute reciprocals
245         do i = 1, m
246             reciprocal(i) = 1.0d0 / real(fib(i))
247         end do
248         !print*, fib
249     end subroutine FibonacciSeries
250
251     subroutine MATRIXMUL(A, B, m, n, p, C)
252         implicit none
253         integer :: m, n, p, i, j, k
254         real*8 :: A(m,n), B(n,p), C(m,p), temp
255
256         do i = 1, m, 1
257             do j = 1, p, 1
258                 temp = 0.0d0
259                 do k = 1, n, 1
260                     temp = temp + A(i,k)*B(k,j)
261                 end do
262                 C(i,j) = temp
263             end do
264         end do
265         !write(1237,*) C
266         end subroutine
267
268         subroutine PTRVR(s, s1, site, vin, rdm)
```

```
269             implicit none
270             integer :: s, s1, s2
271             real*8 :: trace
272             real*8, dimension(0:2**s-1) :: vin
273             real*8, dimension(0:2**s1-1, 0:2**s1-1) :: rdm
274             integer :: ii, i1, a, i0, ia, j1, oo, k1, x1, y1, j, t1,
    t2, z1
275             integer, dimension(0:s1-1) :: site, site2
276             integer, dimension(0:s1-1) :: bin
277             integer, dimension(0:(2**(s-s1))*(2**s1)-1) :: ind
278             s2 = s - s1
279             x1 = 0
280             y1 = 0
281             ind = 0
282
283             do j1 = 2**s1-1, 0, -1
284                 do ii = 0, 2**s-1
285                     a = ii
286                     do i1 = 0, s1-1
287                         i0 = site(i1)
288                         call DTOBONEBIT(a, ia, i0, s)
289                         site2(i1) = ia
290                     end do
291                     call DTOB(j1, bin, s1)
292                     oo = 1
293                     do k1 = 0, s1-1
294                         oo = oo * (bin(k1) - site2(k1))
295                     end do
296                     if (abs(oo) == 1) then
297                         ind(x1) = ii
298                         x1 = x1 + 1
299                     end if
300                 end do
301             end do
302
303             rdm = 0.0d0
304             do t1 = 0, 2**s1-1
305                 do t2 = 0, 2**s1-1
```

```
306                     do z1 = 0, 2**s2-1
307                         rdm(t1, t2) = rdm(t1, t2) + vin(ind((2**s2)
    * t1 + z1)) * vin(ind((2**s2) * t2 + z1))
308                     end do
309                 end do
310             end do
311         end subroutine
312
313         subroutine DTOBONEBIT(m, ia, i0, s)
314             implicit none
315             integer*4 :: s
316             integer :: tt(0:s-1)
317             integer :: m, ia, i0
318             call DTOB(m, tt, s)
319             ia = tt(i0)
320         end subroutine
321
322         subroutine DTOB(m, tt, s)
323             implicit none
324             integer :: s
325             integer, dimension(0:s-1) :: tt
326             integer :: m, k, a2
327             tt = 0
328             a2 = m
329             do k = 0, s-1
330                 tt(s-k-1) = mod(a2, 2)
331                 a2 = a2 / 2
332                 if (a2 == 0) exit
333             end do
334         end subroutine
335
336         subroutine CONVR(s,i0,j0,vin,con)
337             implicit none
338             integer::s,i0,j0 ,site(0:1),i
339             real*8::vin(0:2**s-1),evs
340             real*8,dimension(0:3,0:3)::rdm,MM1,MM2,MM3,MM4
341             INTEGER::INFO, LDA, LWORK, NN,LDVL,LDVR
342             parameter(NN=4,LDA=NN,LWORK=4*NN,LDVL=NN,LDVR=NN)
```

```
343             real*8::WORK(0:LWORK-1),AA2(0:LDA-1,0:NN-1)
344             real*8::VL(0:NN-1,0:NN-1),VR(0:NN-1,0:NN-1)
345             real*8::WR(0:NN-1),WI(0:NN-1)
346             real*8::ev(NN) ,con,c2,x11,const,sigmay(0:3,0:3)
347             sigmay(0,:)=(/0.0,0.0,0.0,-1.0/)
348             sigmay(1,:)=(/0.0,0.0,1.0,0.0/)
349             sigmay(2,:)=(/0.0,1.0,0.0,0.0/)
350             sigmay(3,:)=(/-1.0,0.0,0.0,0.0/)
351             site(0)=i0
352             site(1)=j0
353             call PTRVR(s,2,site,vin,rdm)
354             MM1=matmul(rdm,sigmay)
355             MM2=matmul(rdm,sigmay)
356             MM3=matmul(MM1,MM2)
357             call dgeev('N', 'N', NN, MM3, LDA, WR, &
358             WI, VL, LDVL, VR, LDVR, WORK, LWORK, INFO)
359             do i=1,NN,1
360                 if (WR(i-1) .lt. 0.0000000000001) then
361                     WR(i-1)=0.0d0
362                 end if
363                 ev(i)=WR(i-1)
364             end do
365             do i=2,4
366                 if(ev(i)>=ev(1))then
367                     evs=ev(i)
368                     ev(i)=ev(1)
369                     ev(1)=evs
370                 end if
371             end do
372             do i=3,4
373                 if(ev(i)>=ev(2))then
374                     evs=ev(i)
375                     ev(i)=ev(2)
376                     ev(2)=evs
377                 end if
378             end do
379             do i=4,4
380                 if(ev(i)>=ev(3))then
```

```
                    evs=ev(i)
                    ev(i)=ev(3)
                    ev(3)=evs
                end if
            end do
            c2=0
            con=0
            c2=dsqrt(ev(1))-dsqrt(ev(2))-dsqrt(ev(3))-dsqrt(ev(4))
            if(c2>0)then
                con=c2
            else
                con=0
            end if
        end subroutine
end program program study_of_concurrence_for_Isinglike_Hamiltonian
```

Let us see the variation of possible concurrence for the 3-qubit periodic chain system in Fig.[4.7] as well as 12-qubit periodic chain systems in Fig.[??], which are shown below.

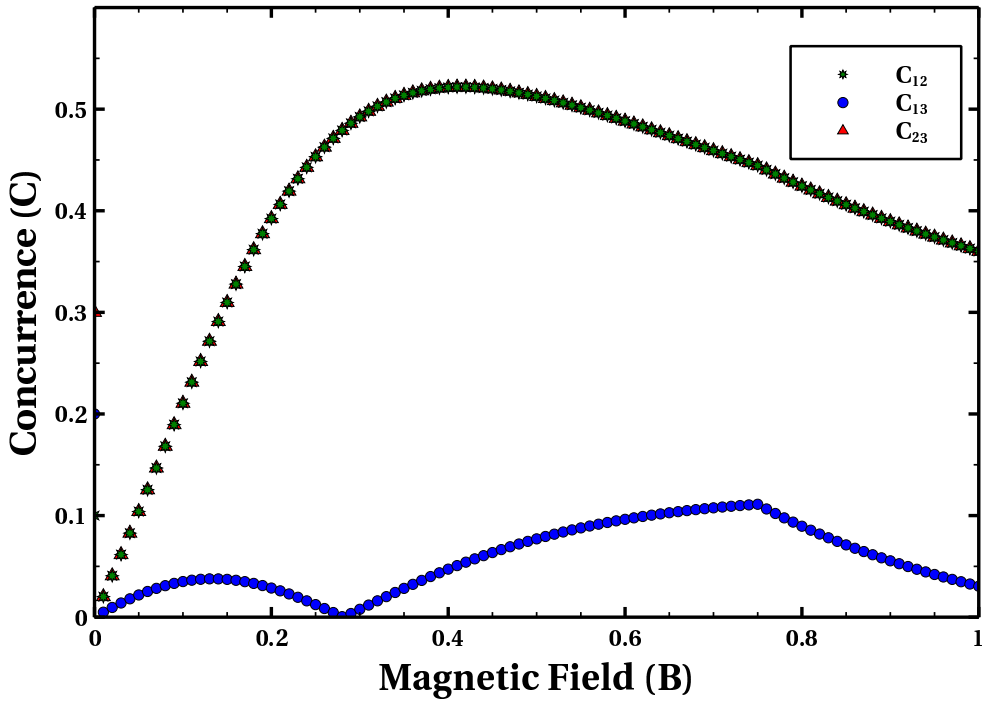
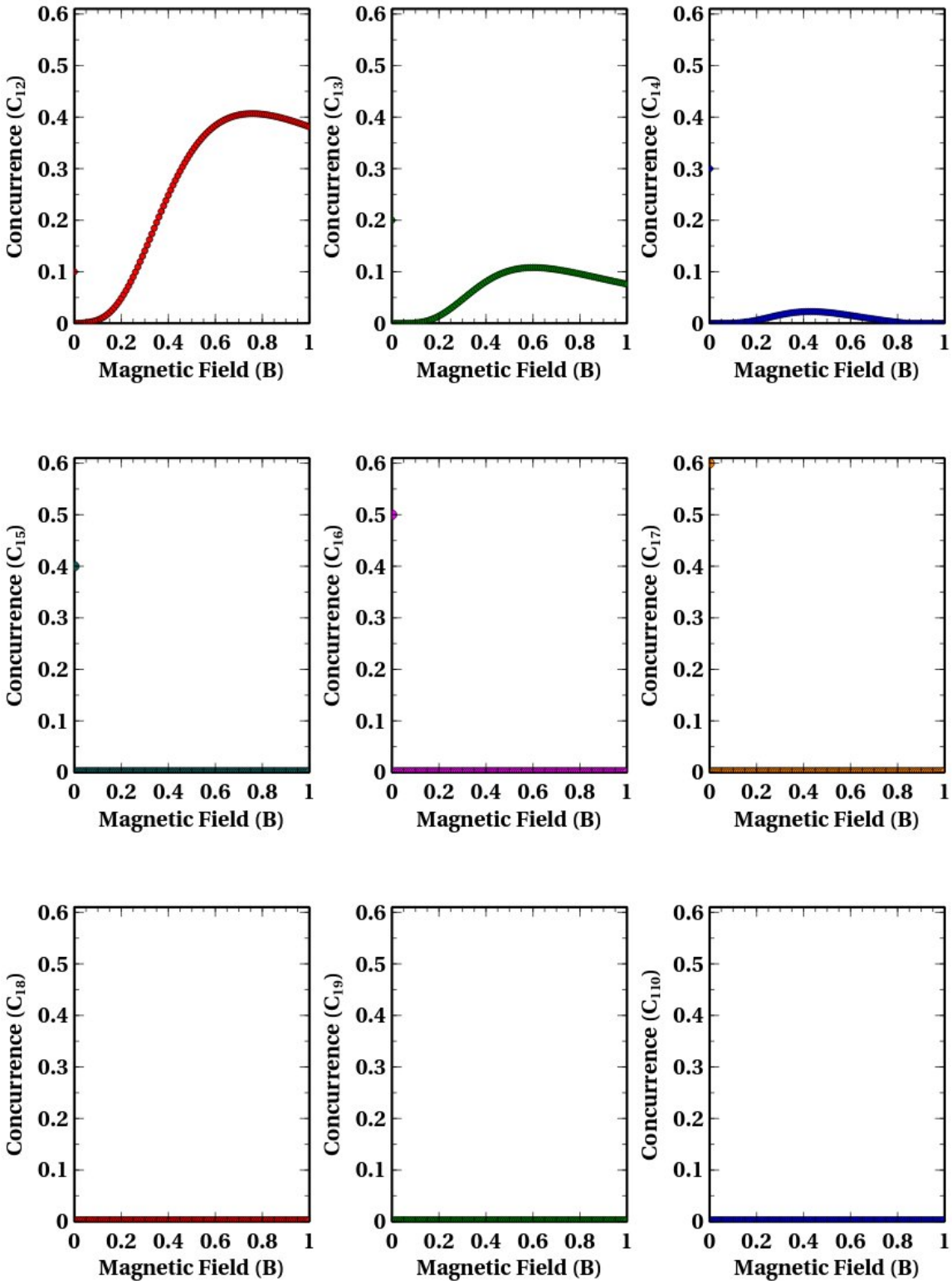
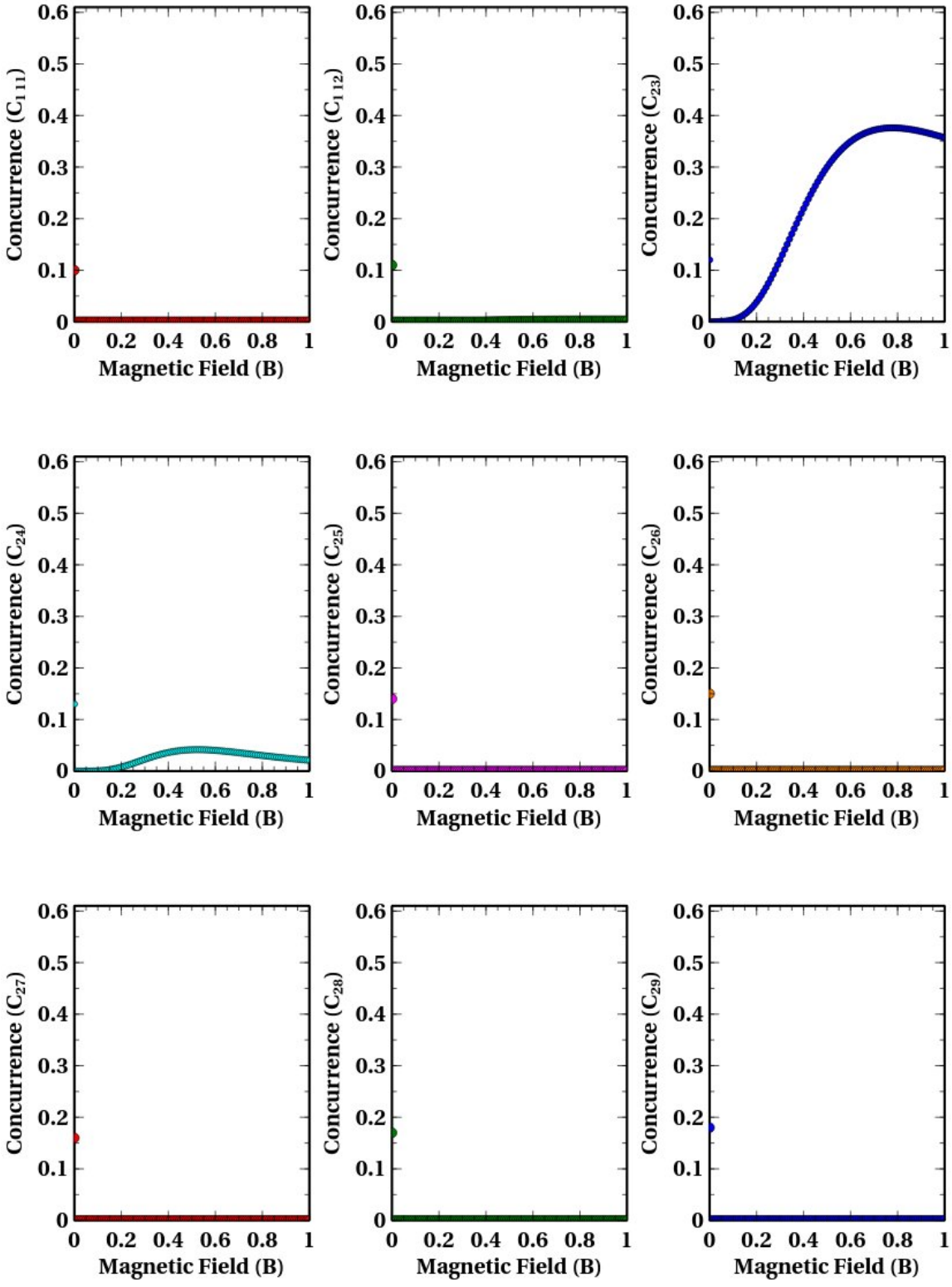


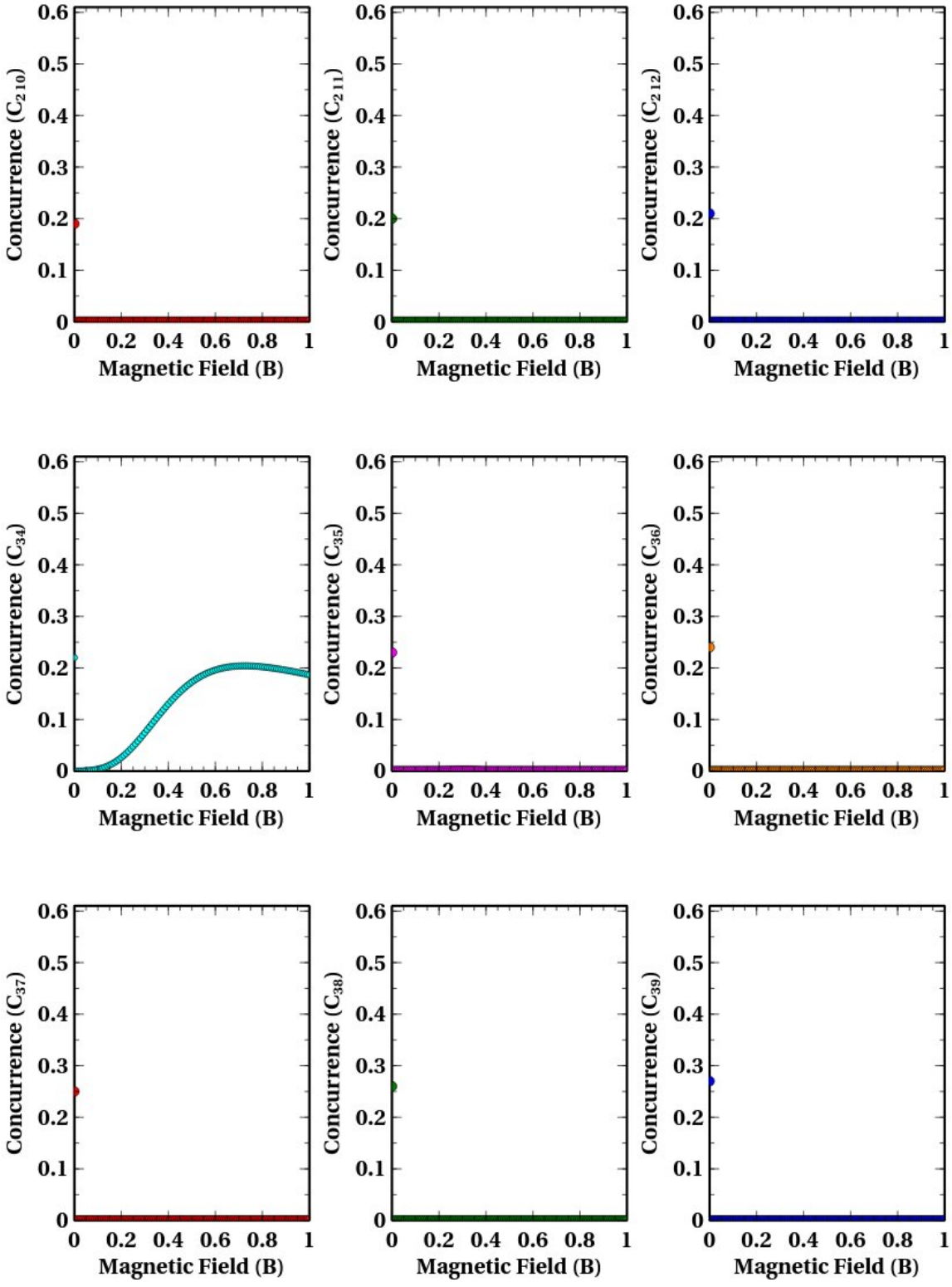
Figure 4.7: Concurrence Study for 3-qubit periodic chain Ising-like System

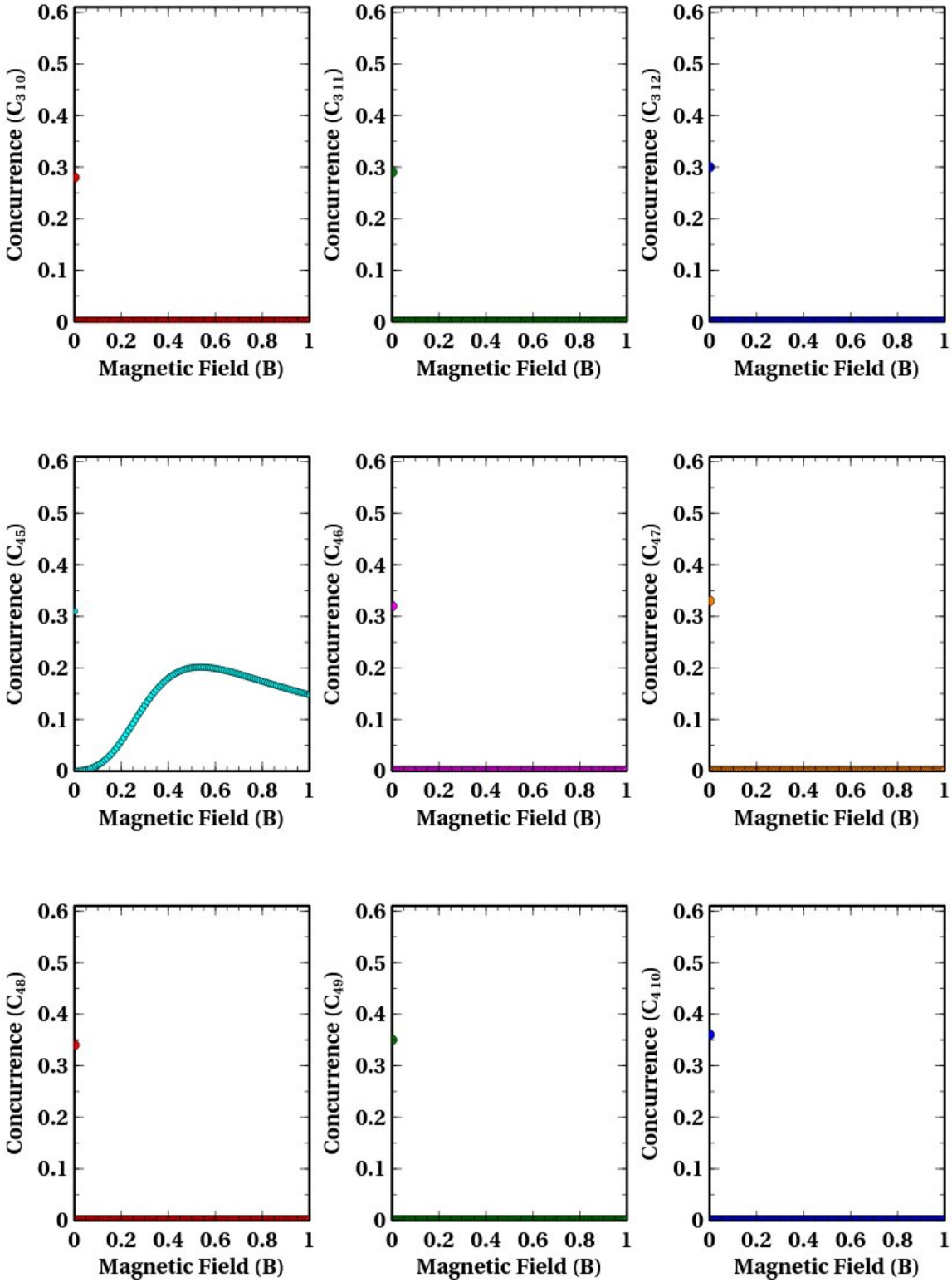
We can see from Fig.[4.7] that the concurrences between the 1st and 2nd qubit (C_{12}) and the 2nd and 3rd qubit (C_{23}) increase rapidly from zero, reach a maximum of 0.5, and then start to decrease. It is interesting to see the variation of concurrence between the 1st and 3rd qubit (C_{13}), which also increases with a magnetic field. Still, it reaches a much smaller value and starts to decrease, reaching a value of 0.00005115 at the magnetic field of 0.28 units. After that, it again starts to rise, but to a much smaller value than C_{12} and C_{23}. Let us discuss the higher qubit system; for example, we can consider the 12-qubit system and show the variation of concurrences between different qubit pairs as follows.

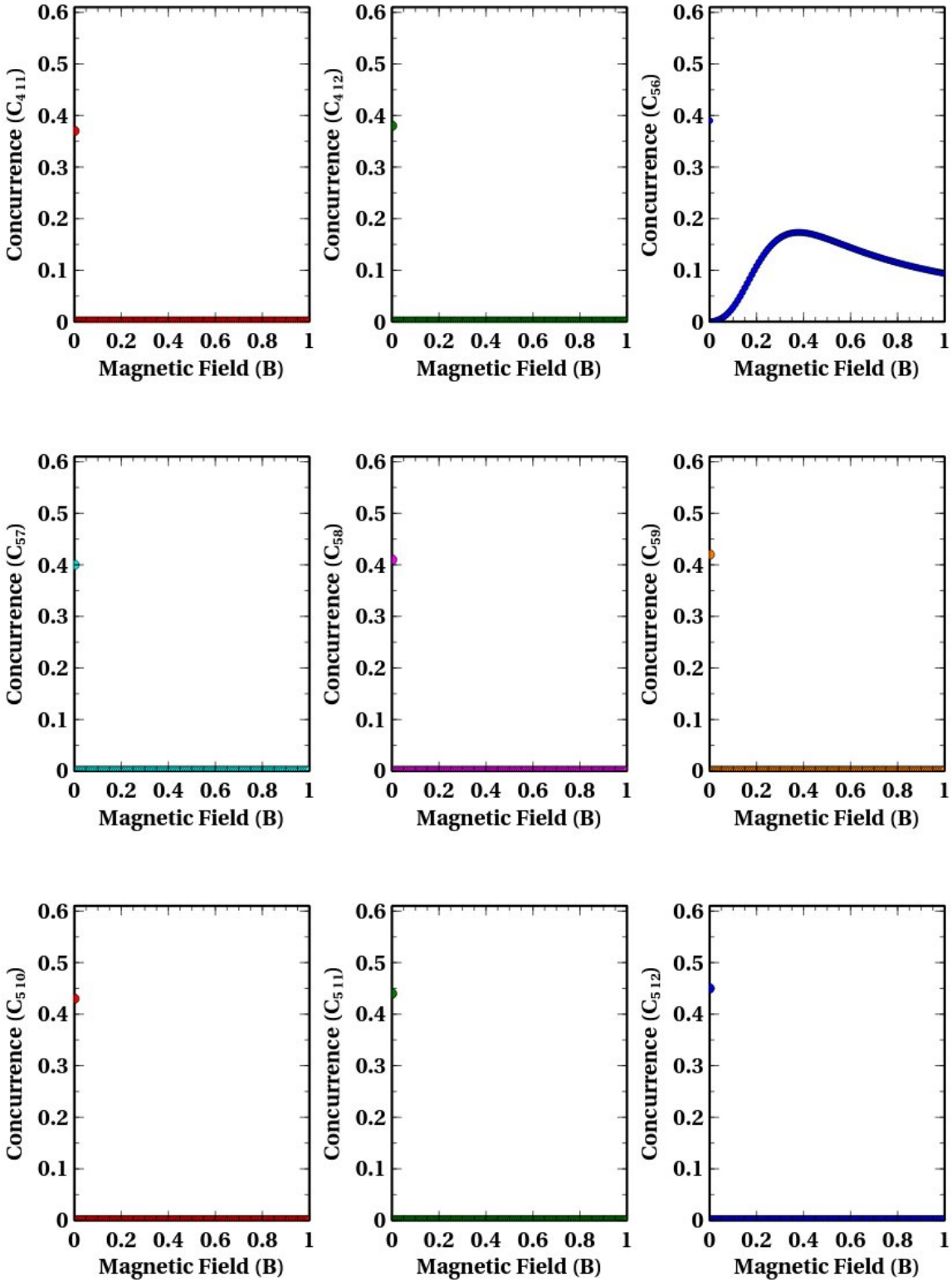
4.3.1.4 Concurrence study for 12 qubit periodic chain Ising-like system

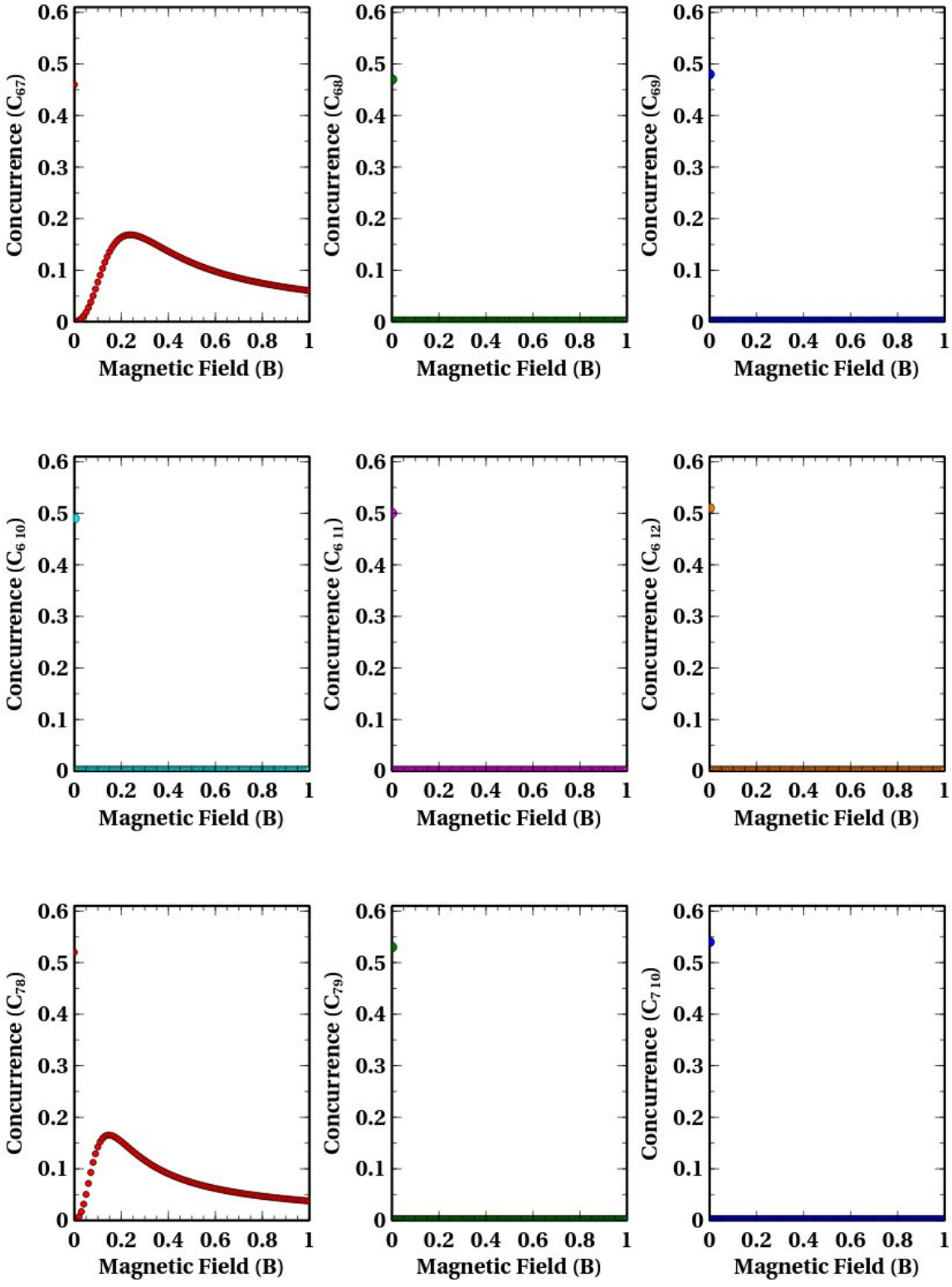


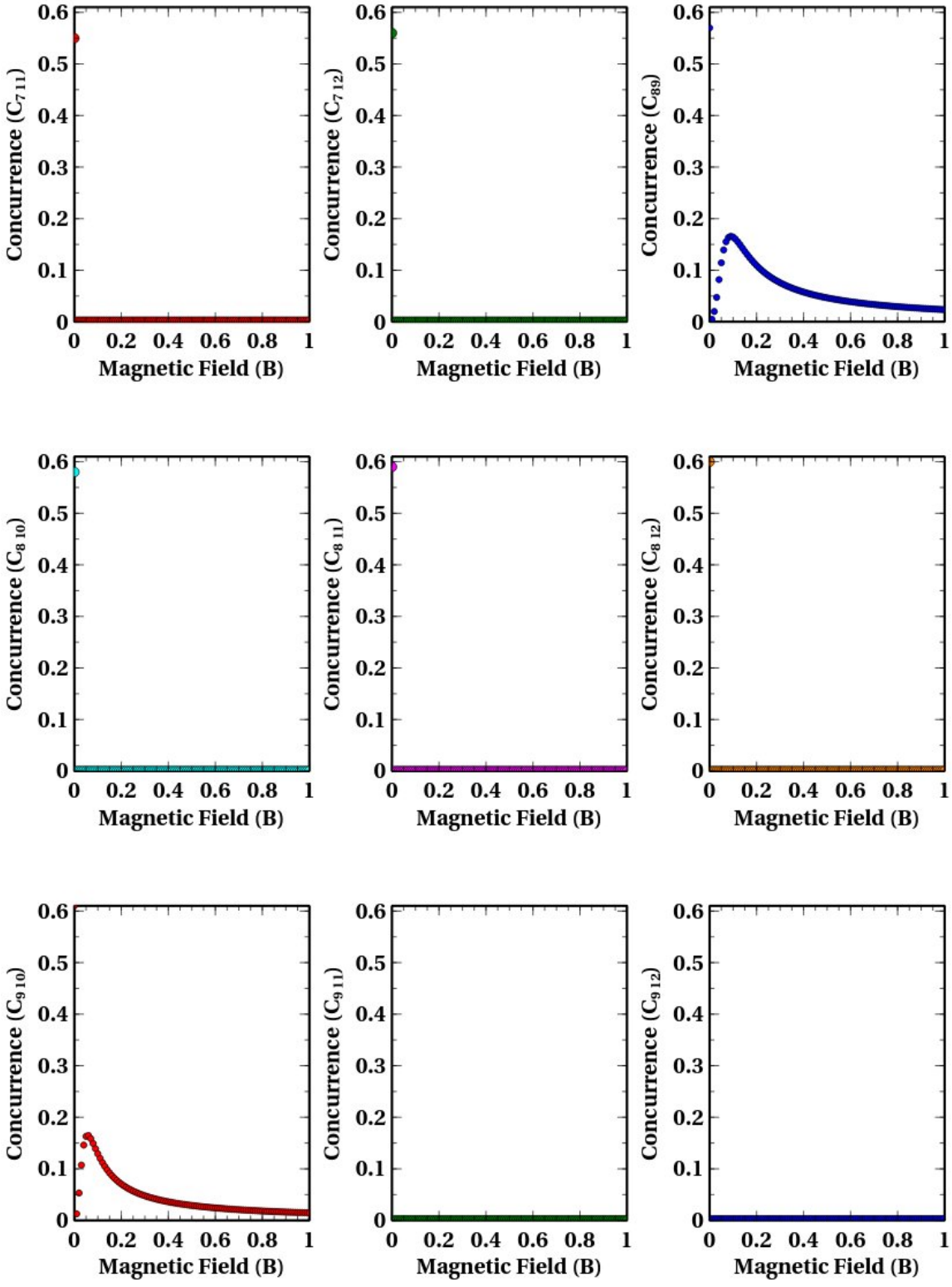

Concurrence ($C_{1\,11}$)
Concurrence ($C_{1\,12}$)
Concurrence (C_{23})
Concurrence (C_{24})
Concurrence (C_{25})
Concurrence (C_{26})
Concurrence (C_{27})
Concurrence (C_{28})
Concurrence (C_{29})
Magnetic Field (B)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.8
1

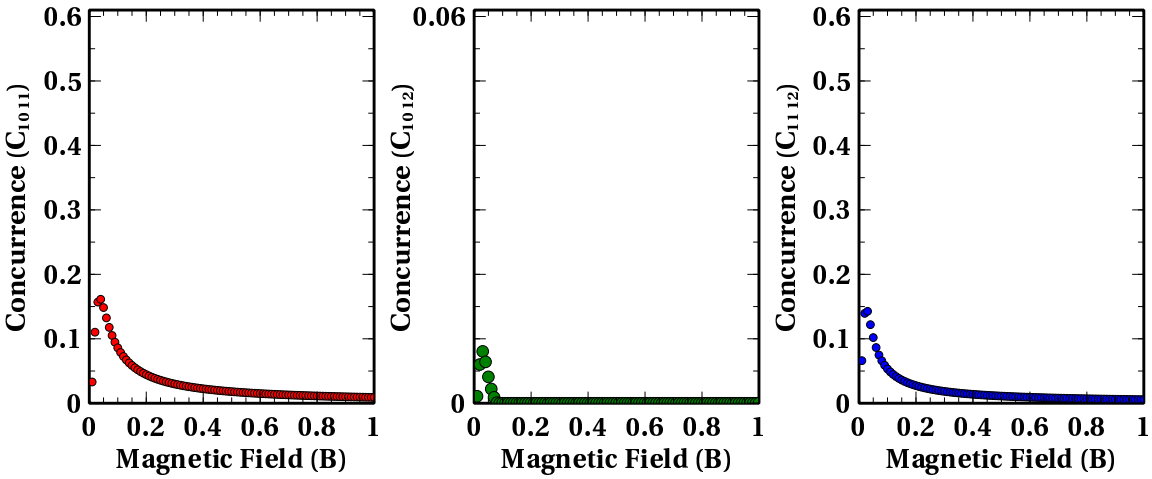

Concurrence (C_{210})
Concurrence (C_{211})
Concurrence (C_{212})
Concurrence (C_{34})
Concurrence (C_{35})
Concurrence (C_{36})
Concurrence (C_{37})
Concurrence (C_{38})
Concurrence (C_{39})
Magnetic Field (B)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.8
1


Concurrence (C_{310})
Concurrence (C_{311})
Concurrence (C_{312})
Concurrence (C_{45})
Concurrence (C_{46})
Concurrence (C_{47})
Concurrence (C_{48})
Concurrence (C_{49})
Concurrence (C_{410})
Magnetic Field (B)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.2
0.4
0.6
0.8
1


Concurrence (C_{411})
Concurrence (C_{412})
Concurrence (C_{56})
Concurrence (C_{57})
Concurrence (C_{58})
Concurrence (C_{59})
Concurrence (C_{510})
Concurrence (C_{511})
Concurrence (C_{512})
Magnetic Field (B)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.2
0.4
0.6
0.8
1


Concurrence (C_{67})
Concurrence (C_{68})
Concurrence (C_{69})
Concurrence ($C_{6\,10}$)
Concurrence ($C_{6\,11}$)
Concurrence ($C_{6\,12}$)
Concurrence (C_{78})
Concurrence (C_{79})
Concurrence ($C_{7\,10}$)
Magnetic Field (B)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.2
0.4
0.6
0.8
1


Concurrence ($C_{7\,11}$)
Concurrence ($C_{7\,12}$)
Concurrence (C_{89})
Concurrence ($C_{8\,10}$)
Concurrence ($C_{8\,11}$)
Concurrence ($C_{8\,12}$)
Concurrence ($C_{9\,10}$)
Concurrence ($C_{9\,11}$)
Concurrence ($C_{9\,12}$)
Magnetic Field (B)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.2
0.4
0.6
0.8
1



For the 12-qubit periodic chain system, the nature are same as can be predicted from the 3-qubit chain system.

4.3.2 Study of Block Entropy for the System

To study the block entropy of the system, we have divided the total system into two blocks: one corresponding to two qubits and another corresponding to the rest of the qubits, as shown in Fig.[4.8].

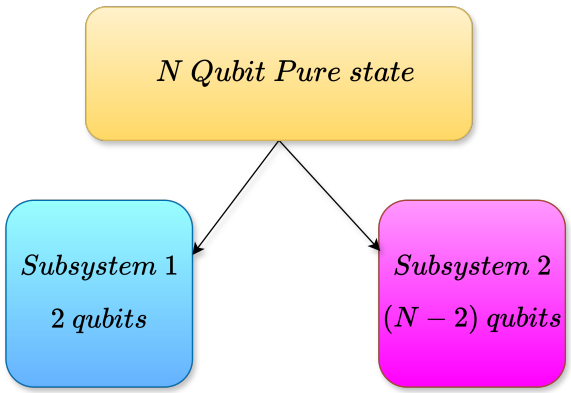


Figure 4.8: Division of the total system into two blocks.

Block entropy is an essential quantity in the study of quantum systems since it helps to analyze the entanglement between the subsystems. By partitioning the

system, we can examine how information is distributed and shared across different segments. Let us consider our two cases to study the block entropy.

4.3.2.1 Case-(1) For Open Chain System

We can implement the following Fortran code to study the block entropy of the open chain system.

```
    program study_of_block_entropy_for_Isinglike_Hamiltonian
    !Program of generalized Isinglike Model, when J is varied like
   inverse Fibonacci series and B, is also varied
    !Open chain System
    implicit none
    integer :: n, i, num, a, x, y, k, differing_bits
    integer, parameter :: t=0, c=1
    real*8 :: B,  con, be1, be2
    integer :: m, count
    character(len=:), allocatable :: state(:)
    integer, allocatable :: decimal(:)
    integer, allocatable :: psi(:)
    double precision, allocatable :: p(:), vec1(:), res1(:), vit(:)
    integer :: j
    real*8, allocatable :: S1(:,:), S2(:,:), H(:,:), H1(:,:), S3
   (:,:),rho(:,:), Ham(:,:),lrho(:,:), res(:,:)
    !For PTRVR
    integer :: s
    integer, parameter :: d = 2
    real*8 :: rdm(0:2**d-1, 0:2**d-1)
    integer, allocatable :: site(:)
    ! Parameters for LAPACK DSYEV
    integer :: lda, info, lwork, lda1, lwork1
    real*8, allocatable :: w(:), work(:), w1(:),  work1(:),w2(:),
   work2(:)
    character*1 :: JOBZ, UPLO
    ! Input values
    print*, "Enter the number of bits you want:"
    read*, n
```

```
27      ! print*, "Enter the value of t:"
28      ! read*, t
29      ! print*, "Enter the value of c:"
30      ! read*, c
31      m = n
32      ! Calculate the number of states
33      num = 2**n
34      s = n
35      ! Allocate arrays after determining the value of num
36      allocate(character(len=n) :: state(num))
37      allocate(integer :: decimal(num))
38      allocate(psi(n))
39      allocate(S1(num, num))
40      allocate(S2(num, num))
41      allocate(S3(num, num))
42      allocate(H(num, num))
43      allocate(H1(num, num))
44      allocate(p(m))
45      allocate(rho(0:num-1,0:num-1))
46      allocate(lrho(0:num-1,0:num-1))
47      allocate(vec1(num))
48      allocate(vit(0:num-1))
49      allocate(res1(num))
50      allocate(Ham(num,num))
51      allocate(site(0:d-1))
52      allocate(res(0:num-1,0:num-1))
53
54      ! LAPACK parameters
55      lda = num
56      lwork = 3 * num - 1
57      lda1 = num-1
58      lwork1 = 3*(num-1)-1
59      allocate(real(8) :: w(lda), work(lwork))
60      allocate(real(8) :: w1(lda), work1(lwork))
61
62      ! Printing the possible states
63      !print*, "The states are:"
64      do i = 0, num - 1
```

```
65          call integer_binary(i, state(i + 1), n)  ! Adjusting index
    for 1-based
66      end do
67
68      !write(722,*) "Decimal equivalents of the states"
69
70      ! Convert binary states to decimal
71      do i = 1, num
72          decimal(i) = btod(state(i), n)
73          !write(722,*) decimal(i)
74      end do
75
76      call FibonacciSeries(m, p)
77
78      ! Perform calculations based on the decimal values for S1
79       do i = 1, num
80          do j = 1, num
81              call single_digit(state(i), psi)
82              if (decimal(i) /= decimal(j)) then
83                  S1(i, j) = 0.0
84               else
85                  S1(i, j) = 0.0
86                  do k = 2, n
87                      S1(i, j) = S1(i, j) + p(k-1)*(1.0 - 2.0 * psi(k
    -1)) * (1.0 - 2.0 * psi(k ))
88                  end do
89                  !S1(i, j) = S1(i, j) + p(n)*(1.0 - 2.0 * psi(n)) *
    (1.0 - 2.0 * psi(1))
90              end if
91          end do
92      end do
93      ! Perform calculations based on the decimal values for S2
94      if (t /= 0) then
95          do i = 1, num
96              do j = 1, num
97                  call single_digit(state(i), psi)
98                  if (decimal(i) /= decimal(j)) then
99                      S2(i, j) = 0.0
```

```
                else
                    S2(i, j) = 0.0
                    do k = 1, n
                        S2(i, j) = S2(i,j) + (1.0 - 2.0 * psi(k))
                    end do
                end if
            end do
        end do
    end if

    ! Construct the S3 matrix (for sigma_x )
    S3 = 0.0
    do i = 1, num
        do j = 1, num
            ! Initialize differing_bits counter
            differing_bits = 0
            do k = 1, n
                ! Compare the k-th bit of the i-th and j-th states
                if (btest(decimal(i), k-1) .neqv. btest(decimal(j),
k-1)) then
                    differing_bits = differing_bits + 1
                end if
            end do
            ! If exactly one bit differs, set S2(i, j) = 1.0
            if (differing_bits == 1) then
                S3(i, j) = 1.0
            else
                S3(i, j) = 0.0
            end if
        end do
    end do

    ! Total Hamiltonian
    do y = 0, 100, 1
        B = y * 0.01
        H = (S1 + B * (t * S2 + c * S3))
        do i = 1, num
            do j = 1, num
```

```
                Ham(i,j) = H(i,j)
            end do
        end do
        ! Eigenvalue and eigenvector determination using DSYEV
        JOBZ = 'V'
        UPLO = 'U'
        call DSYEV(JOBZ, UPLO, num, H, lda, w, work, lwork, info)
        vec1 = H(:, 1)
        do i = 1, num
            vit(i-1) = vec1(i)
        end do
        do i = 1, lda
            write(1234,*) W(I) ! Computation of eigenvalues
        end do
        ! Von Neumann Entropy
        do i = 0, n-2,1
            do j = i+1, n-1, 1
                site(0) = 3
                site(1) = 4
                call BERPVR(n,2,site,vit,be1)
                write(30001,*) be1
            end do
        end do
    end do

    deallocate(state)
    deallocate(decimal)
    deallocate(psi)
    deallocate(S1)
    deallocate(S2)
    deallocate(S3)
    deallocate(H)
    deallocate(H1)
    deallocate(p)
    deallocate(rho)
    deallocate(lrho)
    deallocate(vec1)
    deallocate(site)
```

```
175     deallocate(vit)
176 contains
177     function btod(state, n) result(decimal)
178         implicit none
179         integer, intent(in) :: n
180         character(len=n) :: state
181         integer :: decimal, i
182         decimal = 0
183         do i = 1, n
184             if (state(i:i) == '1') then
185                 decimal = decimal + 2**(n-i)
186             end if
187         end do
188     end function btod
189 
190     subroutine integer_binary(num, binary_string, n)
191         implicit none
192         integer :: num
193         integer, intent(in) :: n
194         integer :: i, temp_num
195         character(len=n), intent(out) :: binary_string
196         temp_num = num
197         binary_string = repeat('0', n)
198         do i = 0, n - 1
199             if (mod(temp_num, 2) == 1) then
200                 binary_string(n-i:n-i) = '1'
201             else
202                 binary_string(n-i:n-i) = '0'
203             end if
204             temp_num = temp_num / 2
205         end do
206     end subroutine integer_binary
207 
208     subroutine single_digit(A, psi)
209         implicit none
210         character(len=*), intent(in) :: A
211         integer, intent(out) :: psi(:)
212         integer :: i
```

```
213         do i = 1, len(A)
214             if (A(i:i) == '0') then
215                 psi(i) = 0
216             else if (A(i:i) == '1') then
217                 psi(i) = 1
218             else
219                 print *, "Invalid input in single_digit subroutine"
220             end if
221         end do
222     end subroutine single_digit
223 
224     subroutine FibonacciSeries(m, reciprocal)
225         implicit none
226         integer :: i
227         integer :: m
228         double precision :: fib(m)
229         double precision :: reciprocal(m)
230         if (m <= 0) then
231             print*, "Number of terms must be greater than 0."
232             stop
233         end if
234         ! Initialize the first two terms
235         fib(1) = 1
236         if (n > 1) then
237             fib(2) = 1
238         end if
239         ! Generate the Fibonacci series
240         do i = 3, m
241             fib(i) = fib(i-1) + fib(i-2)
242         end do
243         ! Compute reciprocals
244         do i = 1, m
245             reciprocal(i) = 1.0d0 / real(fib(i))
246         end do
247         !print*, fib
248     end subroutine FibonacciSeries
249     subroutine MATRIXMUL(A, B, m, n, p, C)
250         implicit none
```

```
251         integer :: m, n, p, i, j, k
252         real*8 :: A(m,n), B(n,p), C(m,p), temp
253 
254         do i = 1, m, 1
255             do j = 1, p, 1
256                 temp = 0.0d0
257                 do k = 1, n, 1
258                     temp = temp + A(i,k)*B(k,j)
259                 end do
260                 C(i,j) = temp
261             end do
262         end do
263         write(1237,*) C
264         end subroutine
265 
266     subroutine PTRVR(s, s1, site, vin, rdm)
267         implicit none
268         integer :: s, s1, s2
269         real*8 :: trace
270         real*8, dimension(0:2**s-1) :: vin
271         real*8, dimension(0:2**s1-1, 0:2**s1-1) :: rdm
272         integer :: ii, i1, a, i0, ia, j1, oo, k1, x1, y1, j, t1, t2,
    z1
273         integer, dimension(0:s1-1) :: site, site2
274         integer, dimension(0:s1-1) :: bin
275         integer, dimension(0:(2**(s-s1))*(2**s1)-1) :: ind
276         s2 = s - s1
277         x1 = 0
278         y1 = 0
279         ind = 0
280         do j1 = 2**s1-1, 0, -1
281             do ii = 0, 2**s-1
282                 a = ii
283                 do i1 = 0, s1-1
284                     i0 = site(i1)
285                     call DTOBONEBIT(a, ia, i0, s)
286                     site2(i1) = ia
287                 end do
```

```
288                 call DTOB(j1, bin, s1)
289                 oo = 1
290                 do k1 = 0, s1-1
291                     oo = oo * (bin(k1) - site2(k1))
292                 end do
293                 if (abs(oo) == 1) then
294                     ind(x1) = ii
295                     x1 = x1 + 1
296                 end if
297             end do
298         end do
299         rdm = 0.0d0
300         do t1 = 0, 2**s1-1
301             do t2 = 0, 2**s1-1
302                 do z1 = 0, 2**s2-1
303                     rdm(t1, t2) = rdm(t1, t2) + vin(ind((2**s2) * t1
    + z1)) * vin(ind((2**s2) * t2 + z1))
304                 end do
305             end do
306         end do
307         end subroutine
308
309     subroutine DTOBONEBIT(m, ia, i0, s)
310         implicit none
311         integer*4 :: s
312         integer :: tt(0:s-1)
313         integer :: m, ia, i0
314         call DTOB(m, tt, s)
315         ia = tt(i0)
316     end subroutine
317
318     subroutine DTOB(m, tt, s)
319         implicit none
320         integer :: s
321         integer, dimension(0:s-1) :: tt
322         integer :: m, k, a2
323         tt = 0
324         a2 = m
```

```
325         do k = 0, s-1
326             tt(s-k-1) = mod(a2, 2)
327             a2 = a2 / 2
328             if (a2 == 0)
329                 exit
330             end if
331         end do
332     end subroutine
333
334     subroutine BERPVR(s,s1,site,vin,be)
335         implicit none
336         integer::s, s1,site(0:s1-1)
337         real*8::be,betemp,vin(0:2**s-1),rdm(0:2**s1-1,0:2**s1-1)
338         integer::i,INFO
339         real*8::WORK(0:3*(2**s1)-1),W(0:2**s1-1)
340         call PTRVR(s,s1,site,vin,rdm)
341         call DSYEV('N','U', 2**s1, rdm, 2**s1, &
342         W, WORK,3*(2**s1)-1,INFO)
343         be=0
344         betemp=0
345         do i=0,2**s1-1,1
346             if (W(i)>0.0000000000001) then
347                 betemp=-(dlog(abs(W(i)))*abs(W(i)))/dlog(2.0d0)
348                 be=be+betemp
349             end if
350         end do
351     end subroutine
352 end program study_of_block_entropy_for_Isinglike_Hamiltonian
```

After implementing the above code, we have plotted the block entropy variation corresponding to the first block consisting of two qubits with the change in the applied magnetic field. The variations of the same for the three-qubit system and twelve-qubit system are shown below in Fig.[4.9] and Fig.[**??**].

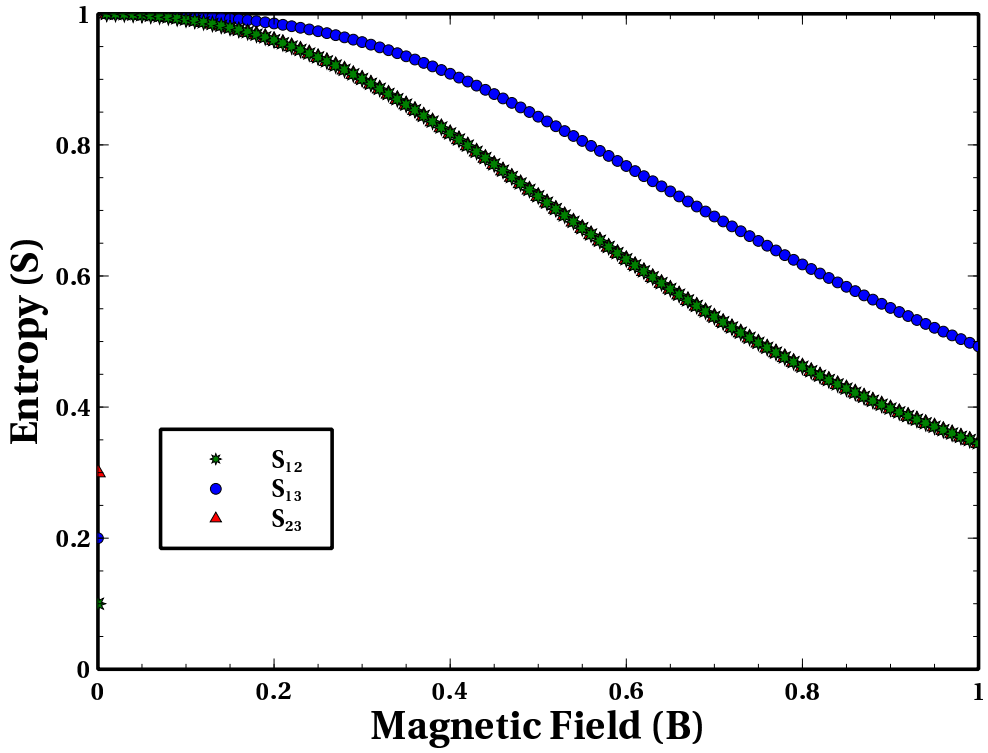
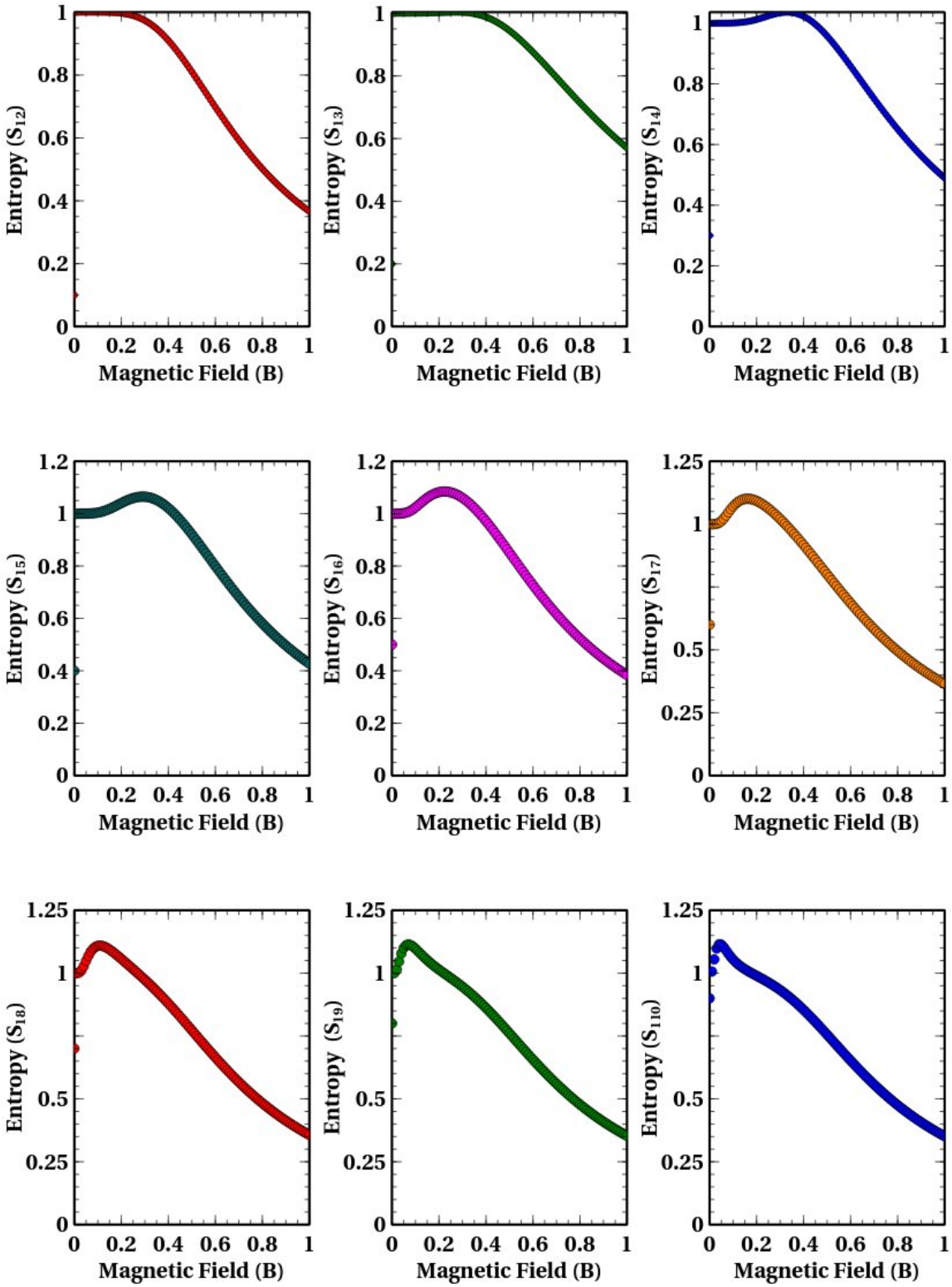
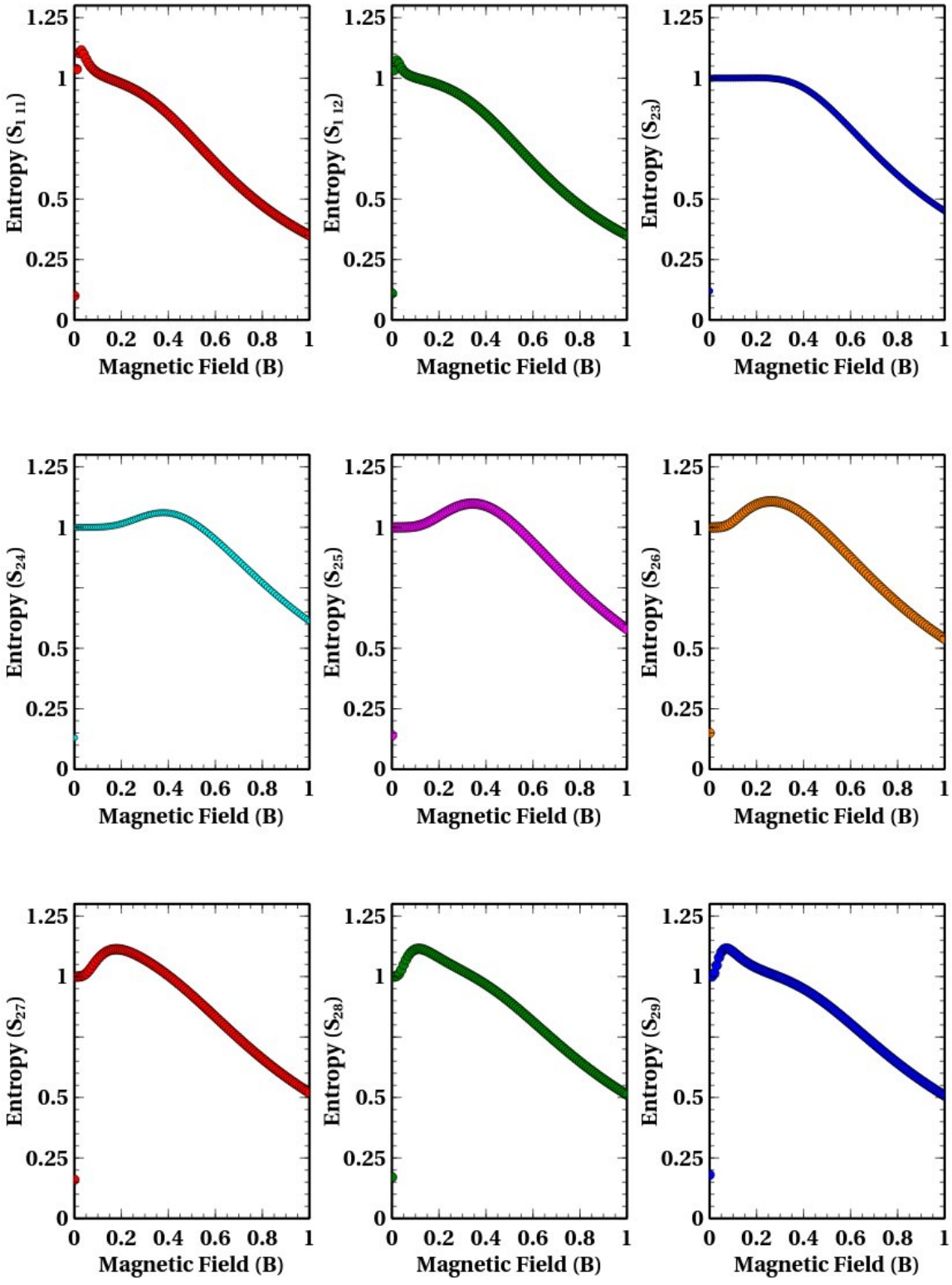


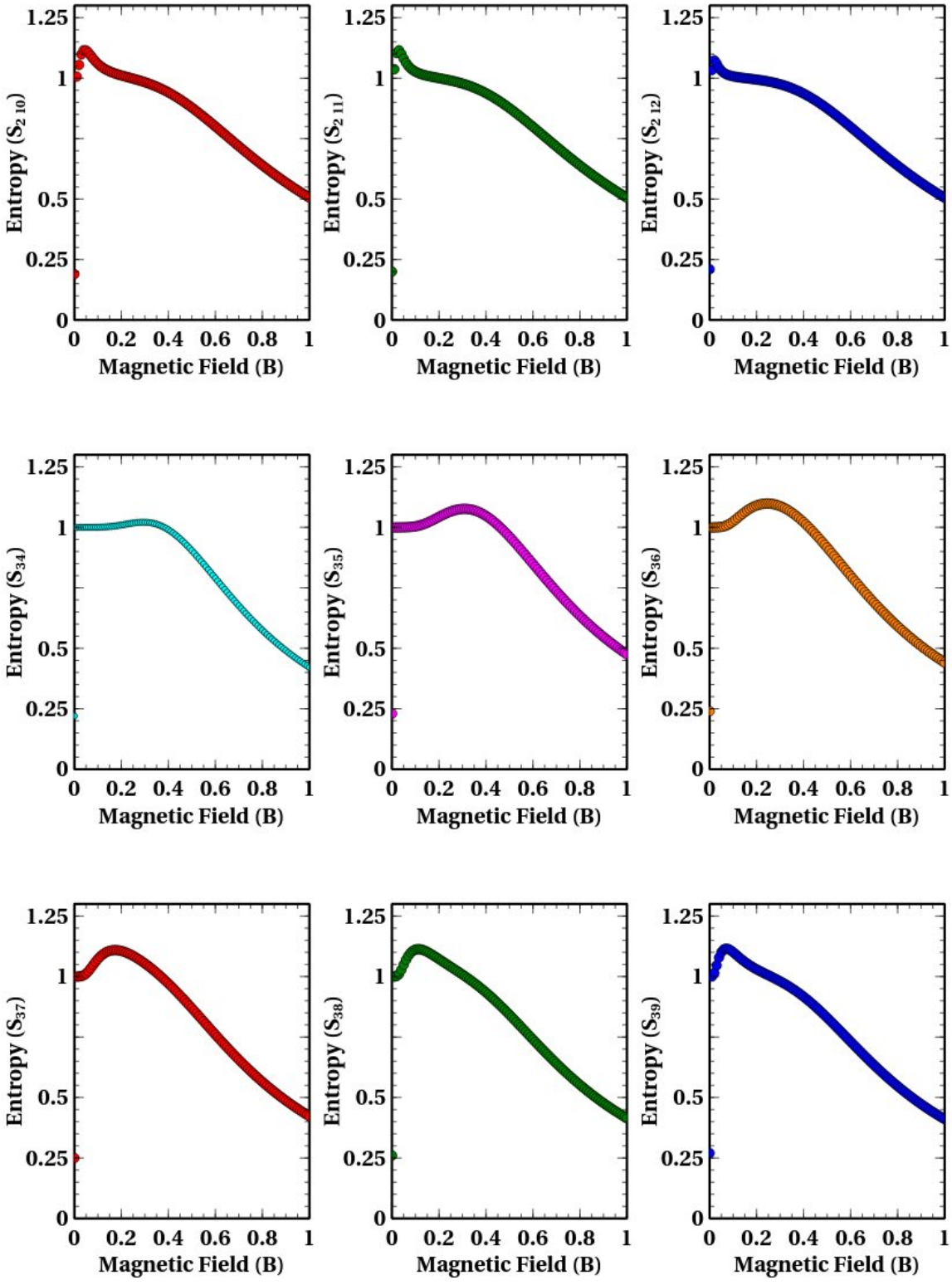
Figure 4.9: Block entropy study for 3 qubits open chain transverse Ising-like system with inverse Fibonacci type of nearest neighbour interaction.

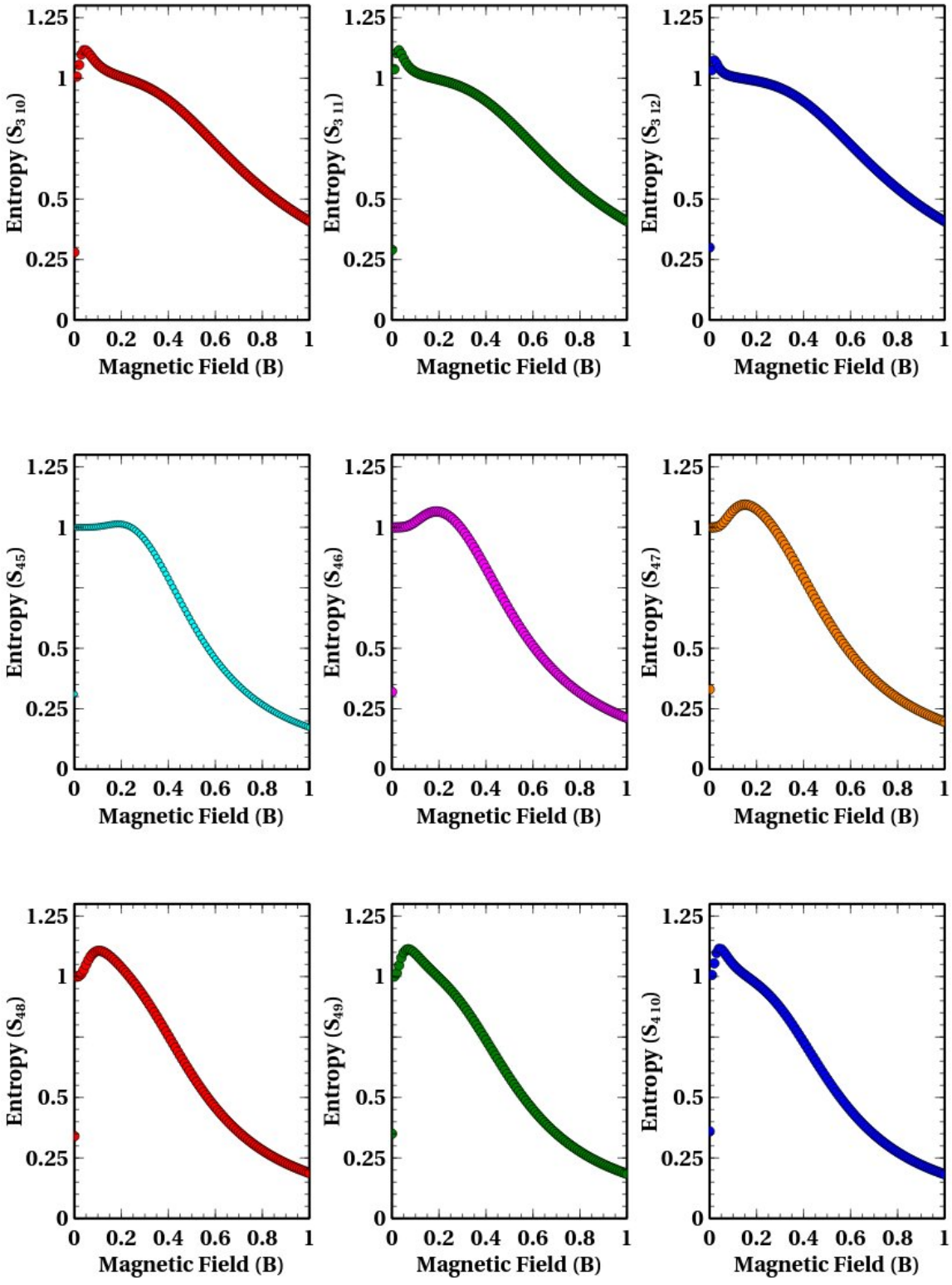
So for a three-qubit open chain system, we can see from Fig.[4.9] that the block entropy corresponding to 1st and 2nd qubit(S_{12}) and 2nd and 3rd qubit(S_{23}) decreases more rapidly than that of 1st and 3rd qubit(S_{13}). Therefore, we can infer that the blocks consisting of 1st, 2nd, and those consisting of 2nd and 3rd are less entangled with the other block that contains only one qubit(remaining one) for the 3-qubit system. Again, the block entropy corresponding to the block containing the 1st and 3rd qubit decreases very slowly, which means the block consisting of these two qubits is more entangled with the block that contains the 2nd qubit. Similarly, for the 12-qubit open chain system, we also get the variation as shown in Fig.[**??**] as below.

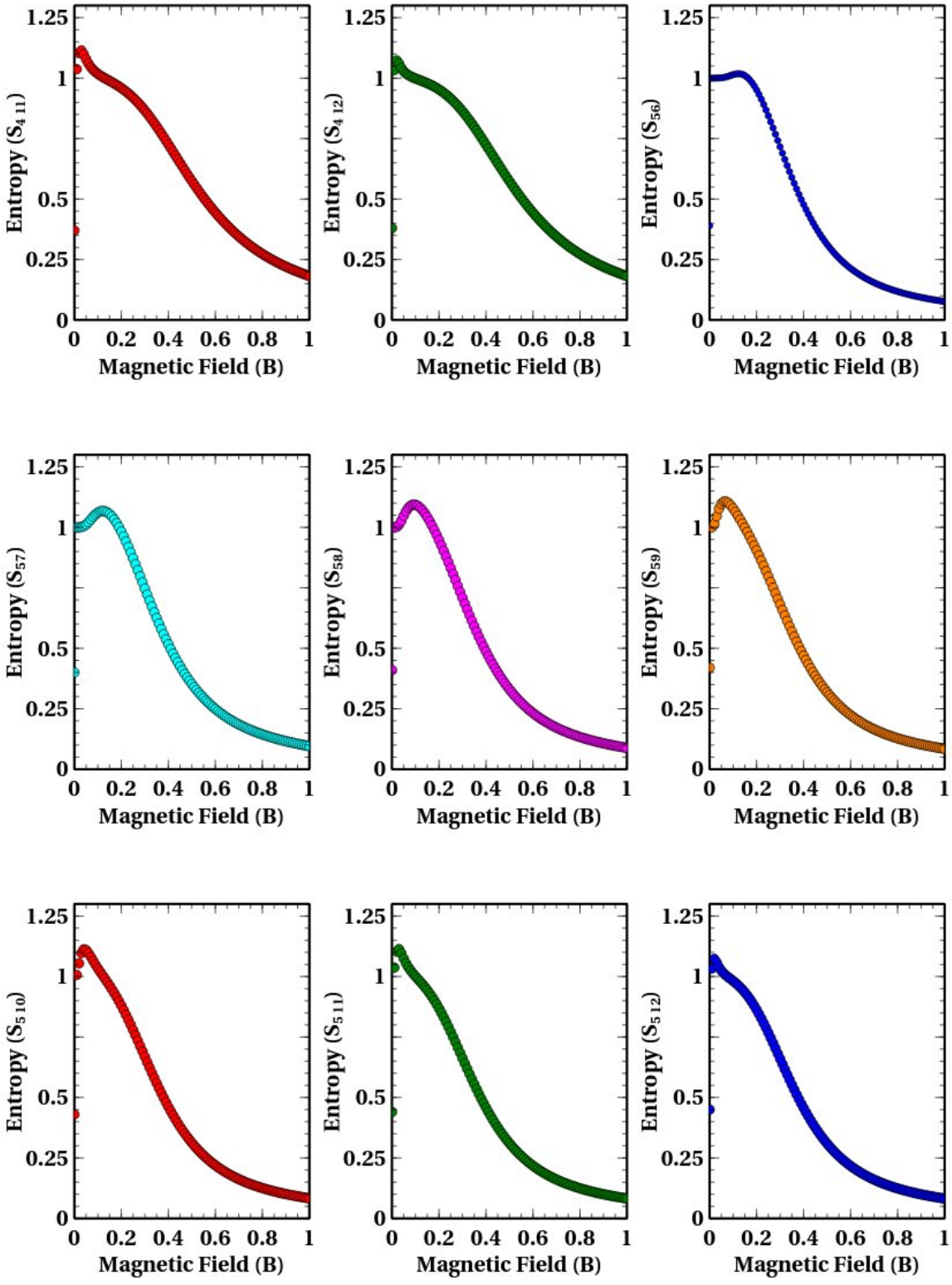
4.3.2.2 Block entropy study for 12 qubit open chain Ising-like system

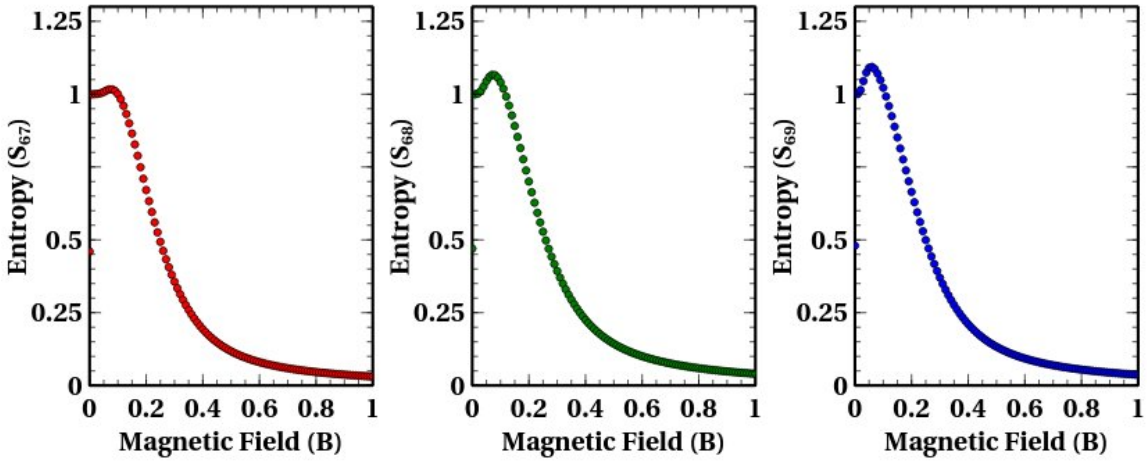


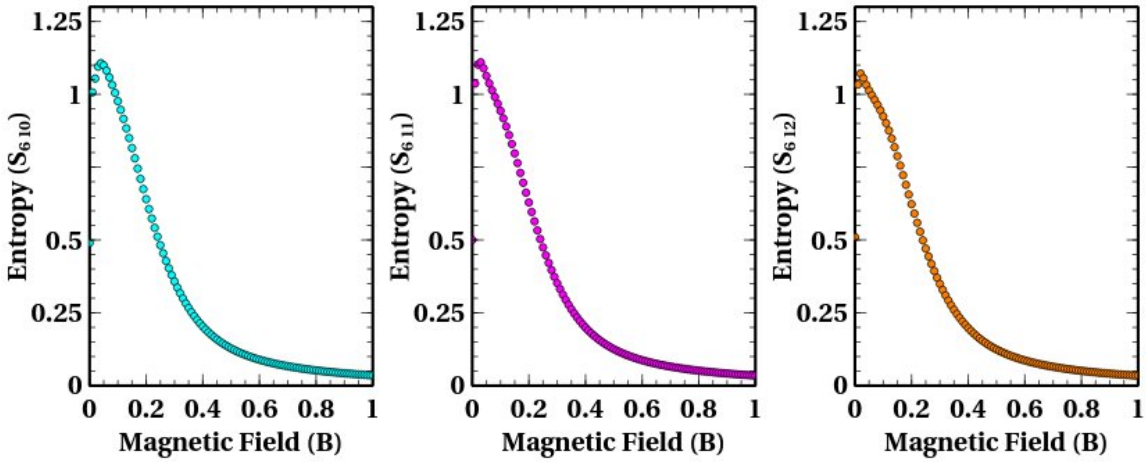

Entropy ($S_{1\,11}$)
Entropy ($S_{1\,12}$)
Entropy (S_{23})
Entropy (S_{24})
Entropy (S_{25})
Entropy (S_{26})
Entropy (S_{27})
Entropy (S_{28})
Entropy (S_{29})
Magnetic Field (B)
0
0.25
0.5
1
1.25
0.2
0.4
0.6
0.8

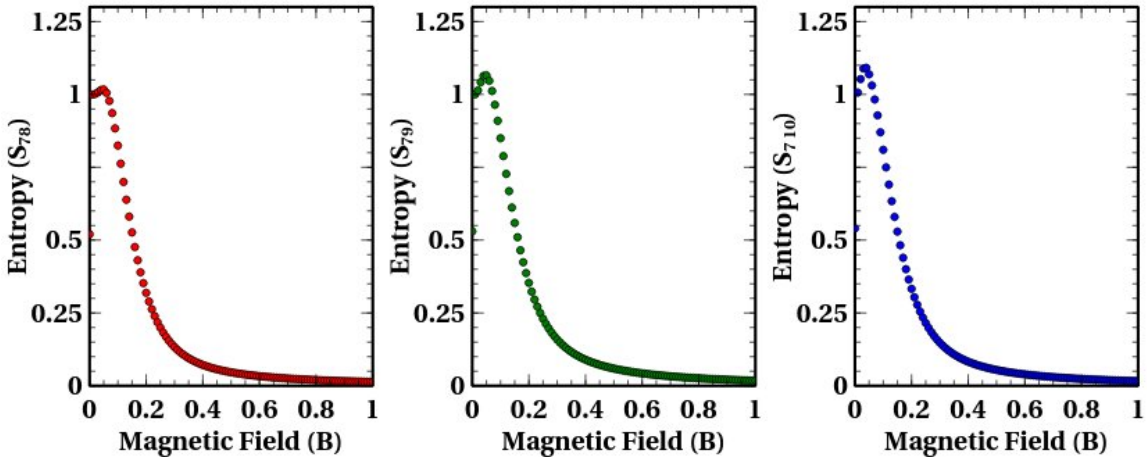

Entropy ($S_{2\,10}$)
Entropy ($S_{2\,11}$)
Entropy ($S_{2\,12}$)
Entropy (S_{34})
Entropy (S_{35})
Entropy (S_{36})
Entropy (S_{37})
Entropy (S_{38})
Entropy (S_{39})
Magnetic Field (B)
0
0.25
0.5
1
1.25
0.2
0.4
0.6
0.8

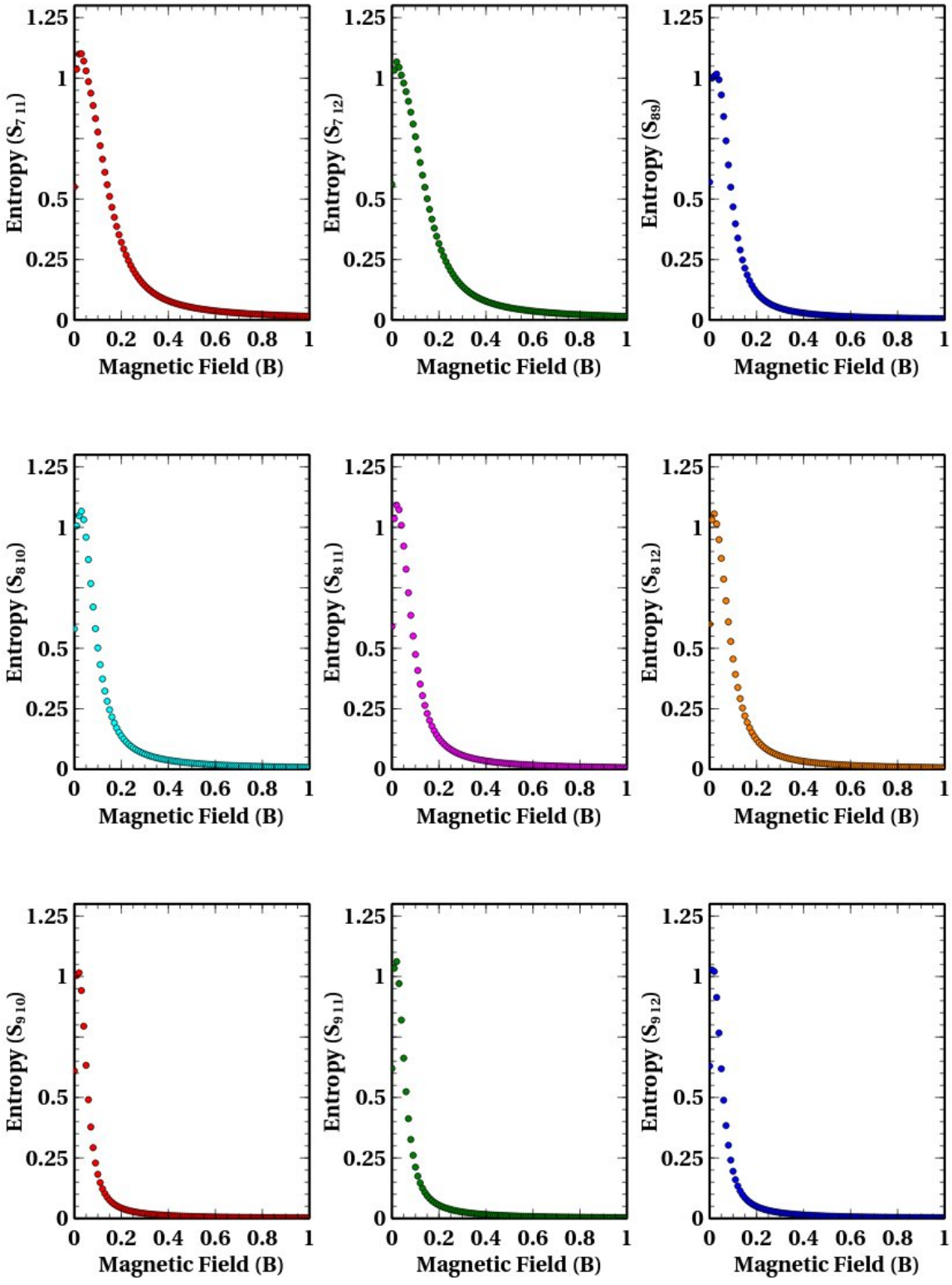

Entropy ($S_{3\,10}$)
Entropy ($S_{3\,11}$)
Entropy ($S_{3\,12}$)
Entropy (S_{45})
Entropy (S_{46})
Entropy (S_{47})
Entropy (S_{48})
Entropy (S_{49})
Entropy ($S_{4\,10}$)
Magnetic Field (B)
0
0.25
0.5
1
1.25
0.2
0.4
0.6
0.8

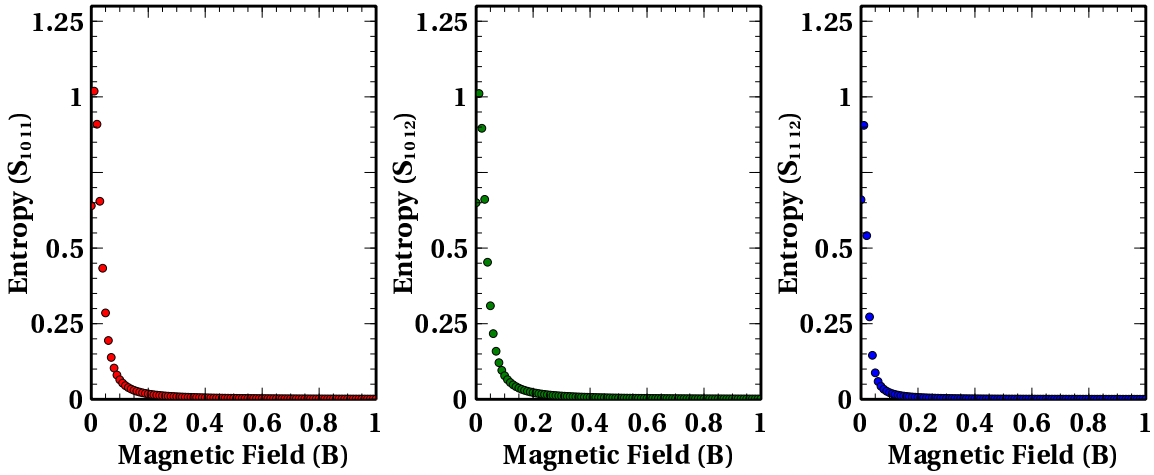

Entropy (S_{411})
Entropy (S_{412})
Entropy (S_{56})
Entropy (S_{57})
Entropy (S_{58})
Entropy (S_{59})
Entropy (S_{510})
Entropy (S_{511})
Entropy (S_{512})
Magnetic Field (B)
0
0.25
0.5
1
1.25
0.2
0.4
0.6
0.8


Entropy (S_{67})
Entropy (S_{68})
Entropy (S_{69})
Magnetic Field (B)
Magnetic Field (B)
Magnetic Field (B)


Entropy ($S_{6\,10}$)
Entropy ($S_{6\,11}$)
Entropy ($S_{6\,12}$)
Magnetic Field (B)
Magnetic Field (B)
Magnetic Field (B)


Entropy (S_{78})
Entropy (S_{79})
Entropy ($S_{7\,10}$)
Magnetic Field (B)
Magnetic Field (B)
Magnetic Field (B)


Entropy ($S_{7\,11}$)
Entropy ($S_{7\,12}$)
Entropy (S_{89})
Entropy ($S_{8\,10}$)
Entropy ($S_{8\,11}$)
Entropy ($S_{8\,12}$)
Entropy ($S_{9\,10}$)
Entropy ($S_{9\,11}$)
Entropy ($S_{9\,12}$)
Magnetic Field (B)
0
0.25
0.5
1
1.25
0.2
0.4
0.6
0.8



4.3.2.3 Case-(2) For Periodic Chain System

Let us now do the study for the periodic chain system by implementing the following code.

```
    program study_of_block_entropy_for_Isinglike_Hamiltonian
    !Program of generalized Isinglike Model, when J is varied like
   inverse Fibonacci series and B, is also varied
    !Periodic chain system
    implicit none
    integer :: n, i, num, a, x, y, k, differing_bits
    integer, parameter :: t=0, c=1
    real*8 :: B,  con, be1, be2
    integer :: m, count
    character(len=:), allocatable :: state(:)
    integer, allocatable :: decimal(:)
    integer, allocatable :: psi(:)
    double precision, allocatable :: p(:), vec1(:), res1(:), vit(:)
    integer :: j
    real*8, allocatable :: S1(:,:), S2(:,:), H(:,:), H1(:,:), S3
   (:,:),rho(:,:), Ham(:,:),lrho(:,:), res(:,:)
    !For PTRVR
    integer :: s
    integer, parameter :: d = 2
    real*8 :: rdm(0:2**d-1, 0:2**d-1)
```

```
19      integer, allocatable :: site(:)
20
21      ! Parameters for LAPACK DSYEV
22      integer :: lda, info, lwork, lda1, lwork1
23      real*8, allocatable :: w(:), work(:), w1(:),  work1(:),w2(:),
    work2(:)
24      character*1 :: JOBZ, UPLO
25
26      ! Input values
27      print*, "Enter the number of bits you want:"
28      read*, n
29      m = n
30      ! Calculate the number of states
31      num = 2**n
32      s = n
33
34      ! Allocate arrays after determining the value of num
35      allocate(character(len=n) :: state(num))
36      allocate(integer :: decimal(num))
37      allocate(psi(n))
38      allocate(S1(num, num))
39      allocate(S2(num, num))
40      allocate(S3(num, num))
41      allocate(H(num, num))
42      allocate(H1(num, num))
43      allocate(p(m))
44      allocate(rho(0:num-1,0:num-1))
45      allocate(lrho(0:num-1,0:num-1))
46      allocate(vec1(num))
47      allocate(vit(0:num-1))
48      allocate(res1(num))
49      allocate(Ham(num,num))
50      allocate(site(0:d-1))
51      allocate(res(0:num-1,0:num-1))
52
53      ! LAPACK parameters
54      lda = num
55      lwork = 3 * num - 1
```

```
    lda1 = num-1
    lwork1 = 3*(num-1)-1
    allocate(real(8) :: w(lda), work(lwork))
    allocate(real(8) :: w1(lda), work1(lwork))

    !print*, "The states are:"
    do i = 0, num - 1
        call integer_binary(i, state(i + 1), n)  ! Adjusting index for 1-based
        !print *, state(i + 1)
    end do

    ! Convert binary states to decimal
    do i = 1, num
        decimal(i) = btod(state(i), n)
        !write(722,*) decimal(i)
    end do

    call FibonacciSeries(m, p)
    ! Perform calculations based on the decimal values for S1
     do i = 1, num
        do j = 1, num
            call single_digit(state(i), psi)
            if (decimal(i) /= decimal(j)) then
                S1(i, j) = 0.0
             else
                S1(i, j) = 0.0
                do k = 1, n - 1
                    S1(i, j) = S1(i, j) + p(k)*(1.0 - 2.0 * psi(k)) * (1.0 - 2.0 * psi(k + 1))
                end do
                S1(i, j) = S1(i, j) + p(n)*(1.0 - 2.0 * psi(n)) * (1.0 - 2.0 * psi(1))
            end if
        end do
    end do
    ! Perform calculations based on the decimal values for S2
    if (t /= 0) then
```

```
        do i = 1, num
            do j = 1, num
                call single_digit(state(i), psi)
                if (decimal(i) /= decimal(j)) then
                    S2(i, j) = 0.0
                else
                    S2(i, j) = 0.0
                    do k = 1, n
                        S2(i, j) = S2(i,j) + (1.0 - 2.0 * psi(k))
                    end do
                end if
            end do
        end do
    end if

    ! Construct the S3 matrix (for sigma_x )
    S3 = 0.0
    do i = 1, num
        do j = 1, num
            ! Initialize differing_bits counter
            differing_bits = 0
            do k = 1, n
                ! Compare the k-th bit of the i-th and j-th states
                if (btest(decimal(i), k-1) .neqv. btest(decimal(j),
k-1)) then
                    differing_bits = differing_bits + 1
                end if
            end do
            ! If exactly one bit differs, set S2(i, j) = 1.0
            if (differing_bits == 1) then
                S3(i, j) = 1.0
            else
                S3(i, j) = 0.0
            end if
        end do
    end do

    ! Total Hamiltonian
```

```
128         do y = 0, 100, 1
129             B = y * 0.01
130             H = (S1 + B * (t * S2 + c * S3))
131             ! write(1222,*)  p(x)
132               do i = 1, num
133                    do j = 1, num
134                      Ham(i,j) = H(i,j)
135                   end do
136                end do
137             ! write(722,*)   Ham
138
139             ! Eigenvalue and eigenvector determination using DSYEV
140             JOBZ = 'V'
141             UPLO = 'U'
142             call DSYEV(JOBZ, UPLO, num, H, lda, w, work, lwork, info
    )
143
144             vec1 = H(:, 1)
145             do i = 1, num
146                 vit(i-1) = vec1(i)
147             end do
148
149             ! Von Neumann Entropy
150             do i = 0, n-2
151                 do j = i+1, n-1, 1
152                     site(0) = i
153                     site(1) = j
154                     call BERPVR(n,2,site,vit,be1)
155                     write(401,*) B, be1
156                 end do
157             end do
158         end do
159
160     deallocate(state)
161     deallocate(decimal)
162     deallocate(psi)
163     deallocate(S1)
164     deallocate(S2)
```

```
    deallocate(S3)
    deallocate(H)
    deallocate(H1)
    deallocate(p)
    deallocate(rho)
    deallocate(lrho)
    deallocate(vec1)
    deallocate(site)
    deallocate(vit)
contains
    function btod(state, n) result(decimal)
        implicit none
        integer, intent(in) :: n
        character(len=n) :: state
        integer :: decimal, i
        decimal = 0
        do i = 1, n
            if (state(i:i) == '1') then
                decimal = decimal + 2**(n-i)
            end if
        end do
    end function btod

    subroutine integer_binary(num, binary_string, n)
        implicit none
        integer :: num
        integer, intent(in) :: n
        integer :: i, temp_num
        character(len=n), intent(out) :: binary_string
        temp_num = num
        binary_string = repeat('0', n)
        do i = 0, n - 1
            if (mod(temp_num, 2) == 1) then
                binary_string(n-i:n-i) = '1'
            else
                binary_string(n-i:n-i) = '0'
            end if
            temp_num = temp_num / 2
```

```
203         end do
204     end subroutine integer_binary
205
206     subroutine single_digit(A, psi)
207         implicit none
208         character(len=*), intent(in) :: A
209         integer, intent(out) :: psi(:)
210         integer :: i
211         do i = 1, len(A)
212             if (A(i:i) == '0') then
213                 psi(i) = 0
214             else if (A(i:i) == '1') then
215                 psi(i) = 1
216             else
217                 print *, "Invalid input in single_digit subroutine"
218             end if
219         end do
220     end subroutine single_digit
221
222     subroutine FibonacciSeries(m, reciprocal)
223         implicit none
224         integer :: i
225         integer :: m
226         double precision :: fib(m)
227         double precision :: reciprocal(m)
228         if (m <= 0) then
229             print*, "Number of terms must be greater than 0."
230             stop
231         end if
232         ! Initialize the first two terms
233         fib(1) = 1
234         if (n > 1) then
235             fib(2) = 1
236         end if
237         ! Generate the Fibonacci series
238         do i = 3, m
239             fib(i) = fib(i-1) + fib(i-2)
240         end do
```

```
241         ! Compute reciprocals
242         do i = 1, m
243             reciprocal(i) = 1.0d0 / real(fib(i))
244         end do
245         !print*, fib
246     end subroutine FibonacciSeries
247     subroutine MATRIXMUL(A, B, m, n, p, C)
248         implicit none
249         integer :: m, n, p, i, j, k
250         real*8 :: A(m,n), B(n,p), C(m,p), temp
251 
252         do i = 1, m, 1
253             do j = 1, p, 1
254                 temp = 0.0d0
255                 do k = 1, n, 1
256                     temp = temp + A(i,k)*B(k,j)
257                 end do
258                 C(i,j) = temp
259             end do
260         end do
261         write(1237,*) C
262         end subroutine
263 
264         subroutine PTRVR(s, s1, site, vin, rdm)
265             implicit none
266             integer :: s, s1, s2
267             real*8 :: trace
268             real*8, dimension(0:2**s-1) :: vin
269             real*8, dimension(0:2**s1-1, 0:2**s1-1) :: rdm
270             integer :: ii, i1, a, i0, ia, j1, oo, k1, x1, y1, j, t1,
    t2, z1
271             integer, dimension(0:s1-1) :: site, site2
272             integer, dimension(0:s1-1) :: bin
273             integer, dimension(0:(2**(s-s1))*(2**s1)-1) :: ind
274             s2 = s - s1
275             x1 = 0
276             y1 = 0
277             ind = 0
```

```
278
279             do j1 = 2**s1-1, 0, -1
280                 do ii = 0, 2**s-1
281                     a = ii
282                     do i1 = 0, s1-1
283                         i0 = site(i1)
284                         call DTOBONEBIT(a, ia, i0, s)
285                         site2(i1) = ia
286                     end do
287                     call DTOB(j1, bin, s1)
288                     oo = 1
289                     do k1 = 0, s1-1
290                         oo = oo * (bin(k1) - site2(k1))
291                     end do
292                     if (abs(oo) == 1) then
293                         ind(x1) = ii
294                         x1 = x1 + 1
295                     end if
296                 end do
297             end do
298
299             rdm = 0.0d0
300             do t1 = 0, 2**s1-1
301                 do t2 = 0, 2**s1-1
302                     do z1 = 0, 2**s2-1
303                         rdm(t1, t2) = rdm(t1, t2) + vin(ind((2**s2)
    * t1 + z1)) * vin(ind((2**s2) * t2 + z1))
304                     end do
305                 end do
306             end do
307         end subroutine
308
309         subroutine DTOBONEBIT(m, ia, i0, s)
310             implicit none
311             integer*4 :: s
312             integer :: tt(0:s-1)
313             integer :: m, ia, i0
314             call DTOB(m, tt, s)
```

```
315             ia = tt(i0)
316         end subroutine
317
318         subroutine DTOB(m, tt, s)
319             implicit none
320             integer :: s
321             integer, dimension(0:s-1) :: tt
322             integer :: m, k, a2
323             tt = 0
324             a2 = m
325             do k = 0, s-1
326                 tt(s-k-1) = mod(a2, 2)
327                 a2 = a2 / 2
328                 if (a2 == 0) exit
329             end do
330         end subroutine
331
332         subroutine BERPVR(s,s1,site,vin,be)
333             implicit none
334             integer::s, s1,site(0:s1-1)
335             real*8::be,betemp,vin(0:2**s-1),rdm(0:2**s1-1,0:2**s1-1)
336             integer::i,INFO
337             real*8::WORK(0:3*(2**s1)-1),W(0:2**s1-1)
338             call PTRVR(s,s1,site,vin,rdm)
339             call DSYEV('N','U', 2**s1, rdm, 2**s1, &
340             W, WORK,3*(2**s1)-1,INFO)
341             be=0
342             betemp=0
343             do i=0,2**s1-1,1
344             if (W(i)>0.0000000000001) then
345             betemp=-(dlog(abs(W(i)))*abs(W(i)))/dlog(2.0d0)
346             be=be+betemp
347             end if
348             end do
349             end subroutine
350 end program study_of_block_entropy_for_Isinglike_Hamiltonian
```

By implementing the above code, we can get the variation of Block entropy as follows in Fig.[4.10] and Fig.[??] for 3-qubit and 12-qubit periodic chain systems, respectively.

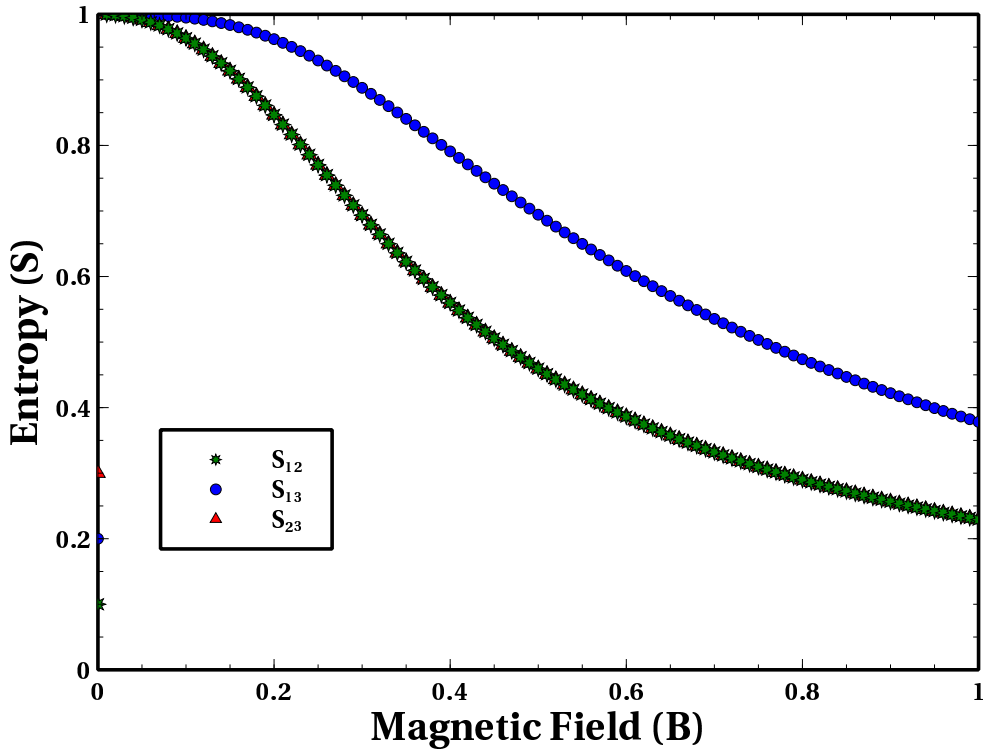
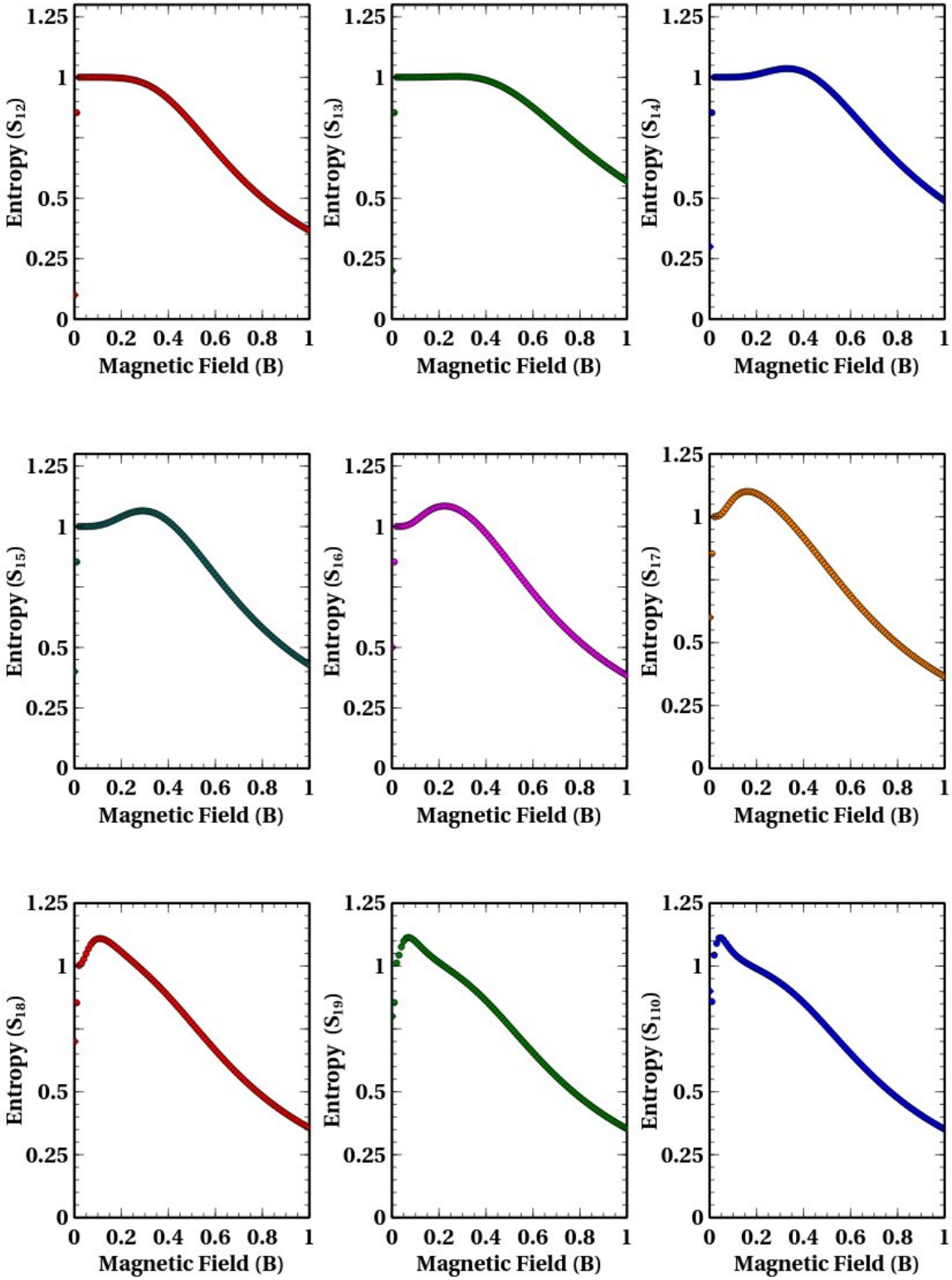
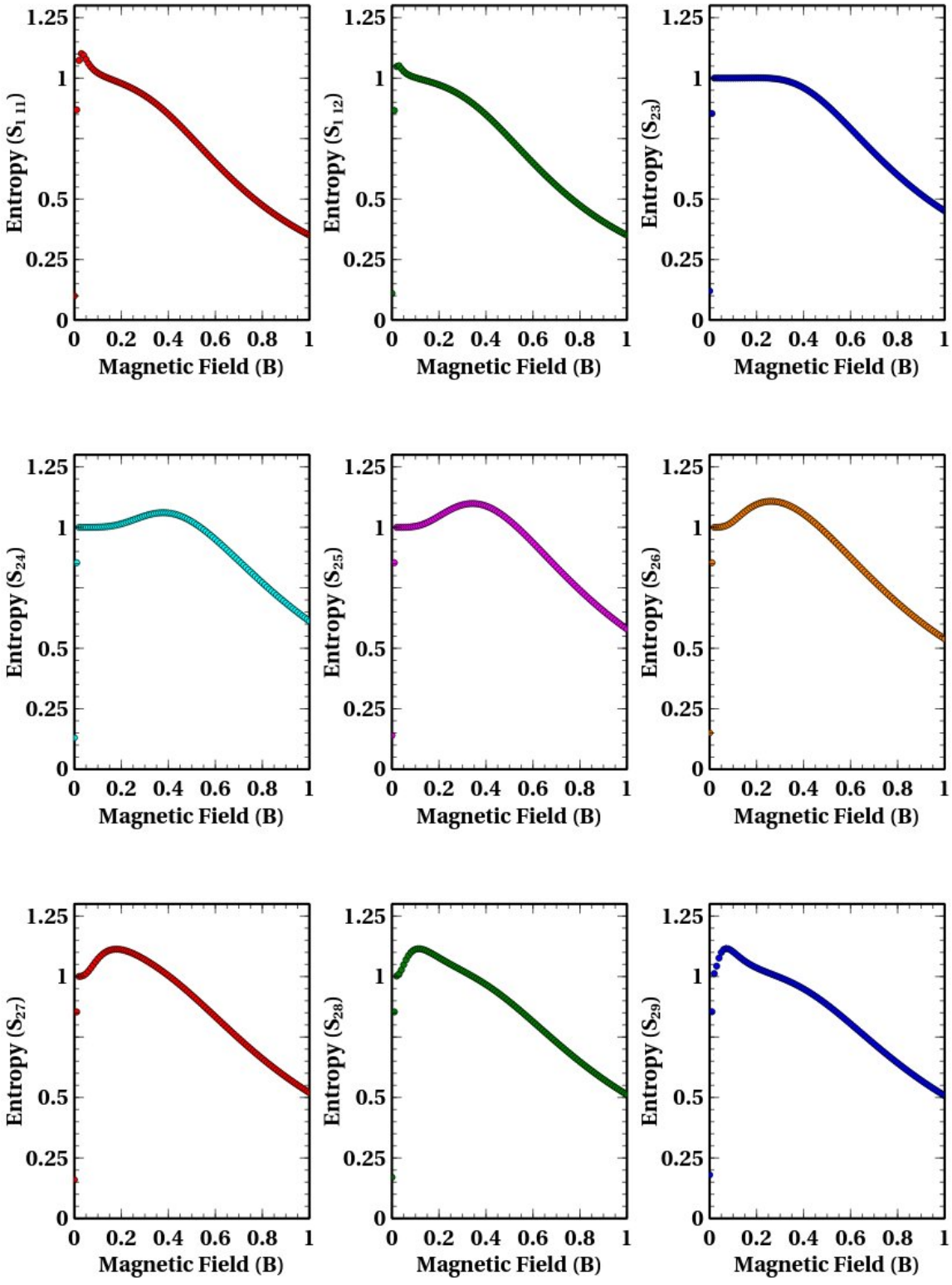


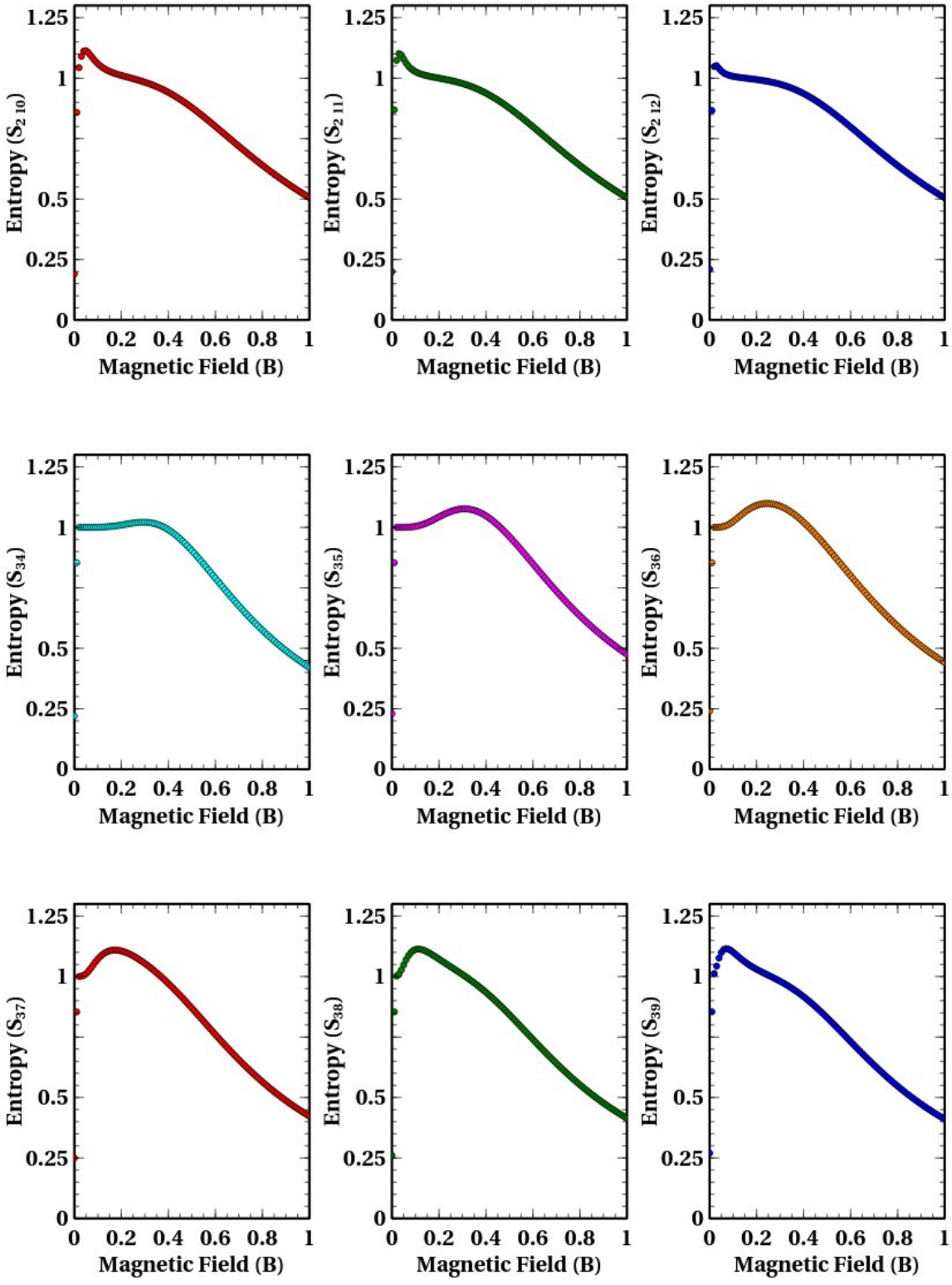
Figure 4.10: Block entropy Study for 3 qubit open chain system

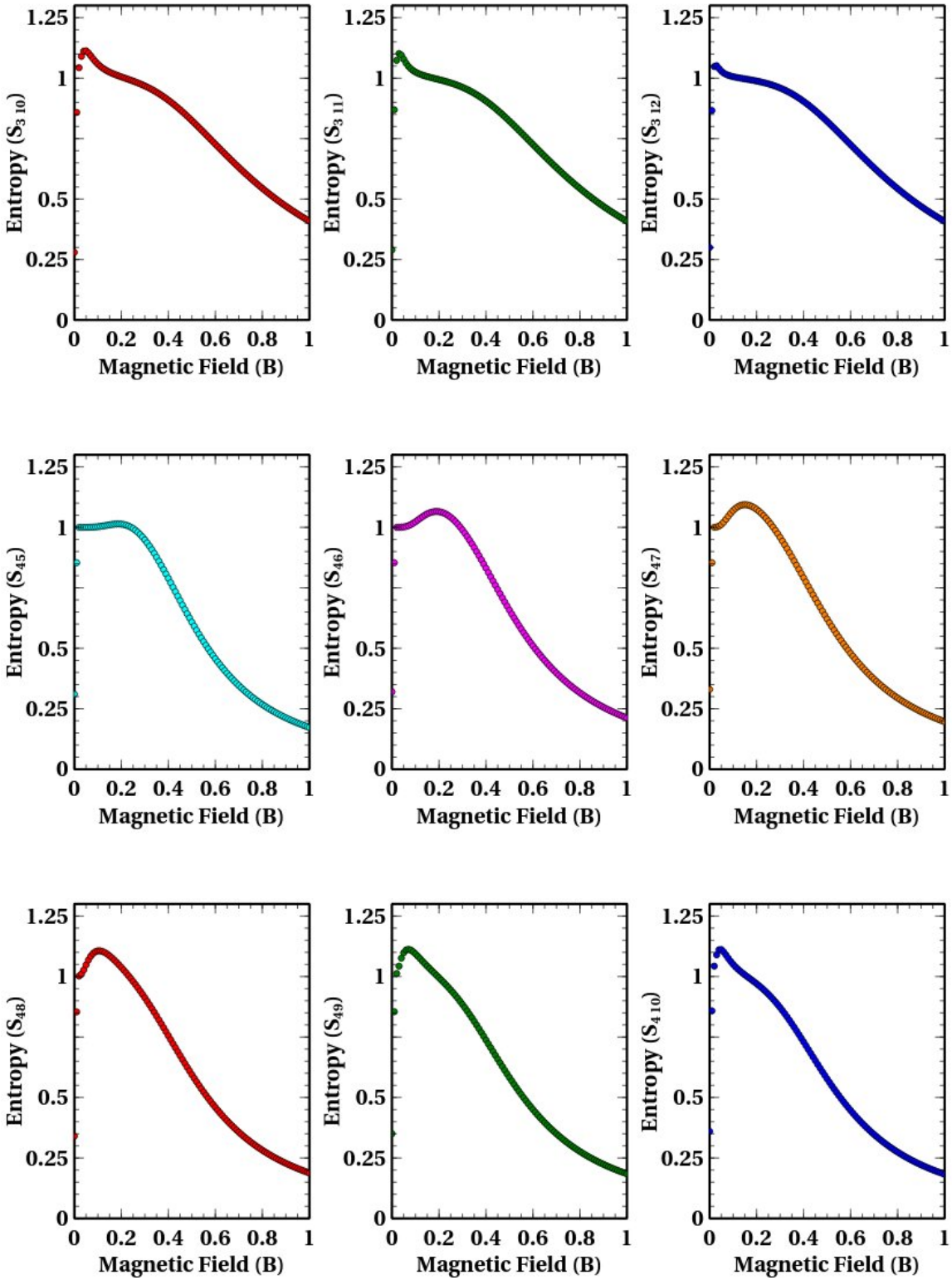
In a three-qubit periodic chain system, we also get the same decreasing variation of S_{12}, S_{23}, and S_{13} as that observed in the open chain system, but the decrease is very rapid in this case. Therefore, we can conclude that the entanglement between the block containing this pair of qubits with the other block containing the respective remaining qubits is more than that for an open chain system.

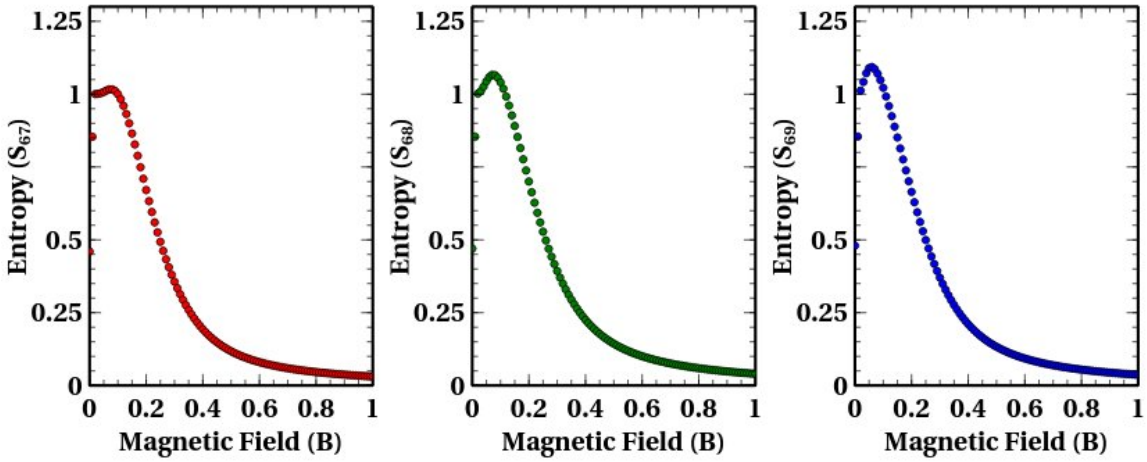
4.3.2.4 Block entropy study for 12 qubit periodic chain Ising-like system

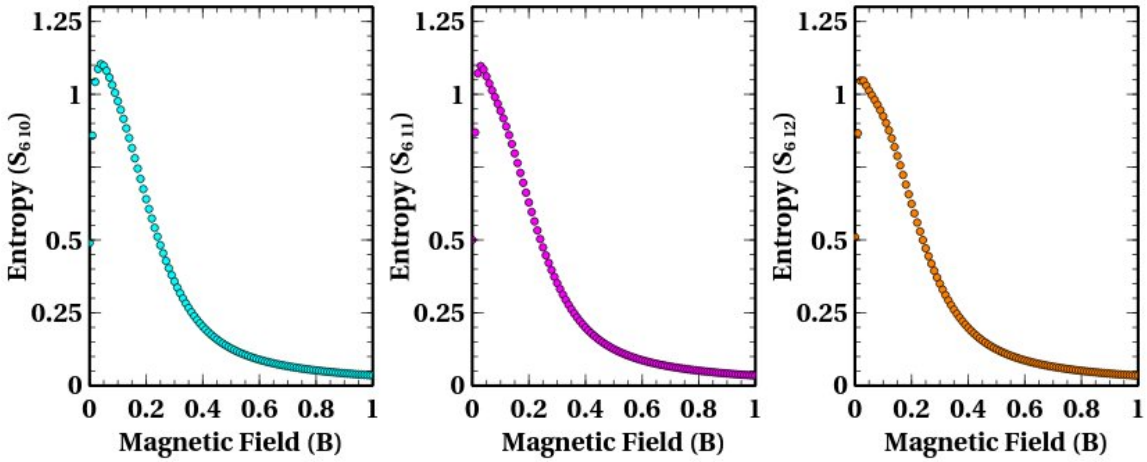


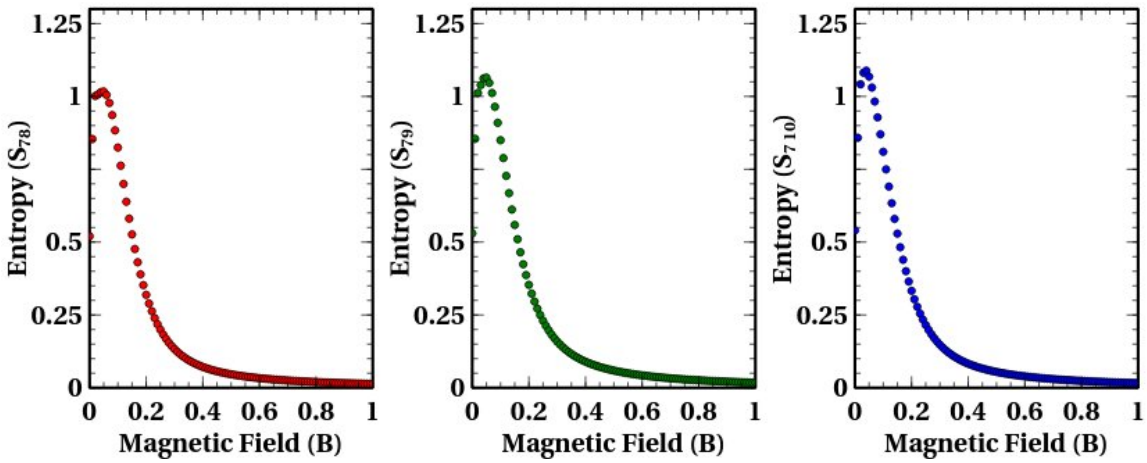

Entropy ($S_{I\ 11}$)
Entropy ($S_{I\ 12}$)
Entropy (S_{23})
Entropy (S_{24})
Entropy (S_{25})
Entropy (S_{26})
Entropy (S_{27})
Entropy (S_{28})
Entropy (S_{29})
Magnetic Field (B)
0
0.25
0.5
1
1.25
0.2
0.4
0.6
0.8

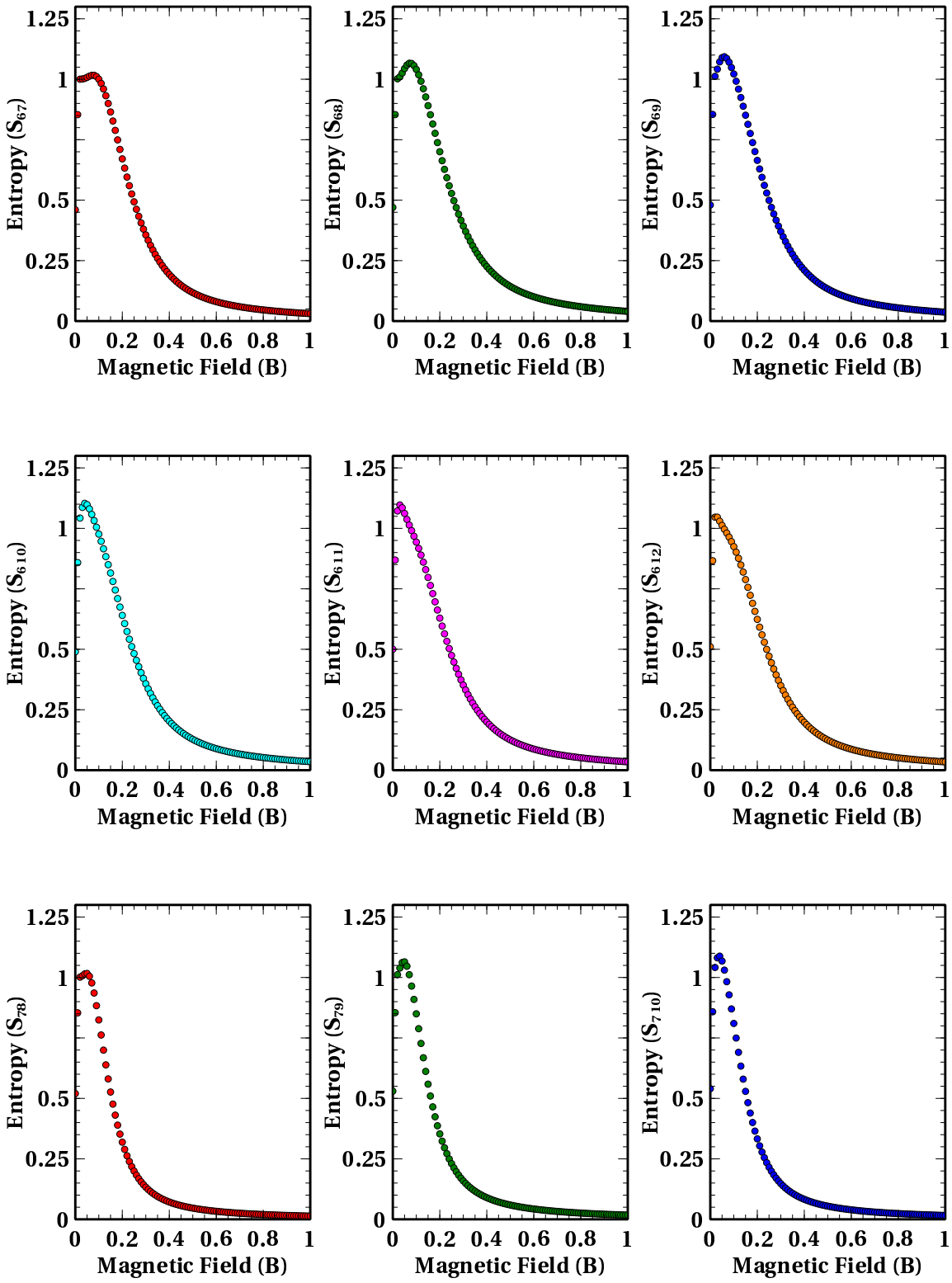

Entropy ($S_{2\,10}$)
Entropy ($S_{2\,11}$)
Entropy ($S_{2\,12}$)
Entropy (S_{34})
Entropy (S_{35})
Entropy (S_{36})
Entropy (S_{37})
Entropy (S_{38})
Entropy (S_{39})
Magnetic Field (B)
0
0.25
0.5
1
1.25
0.2
0.4
0.6
0.8

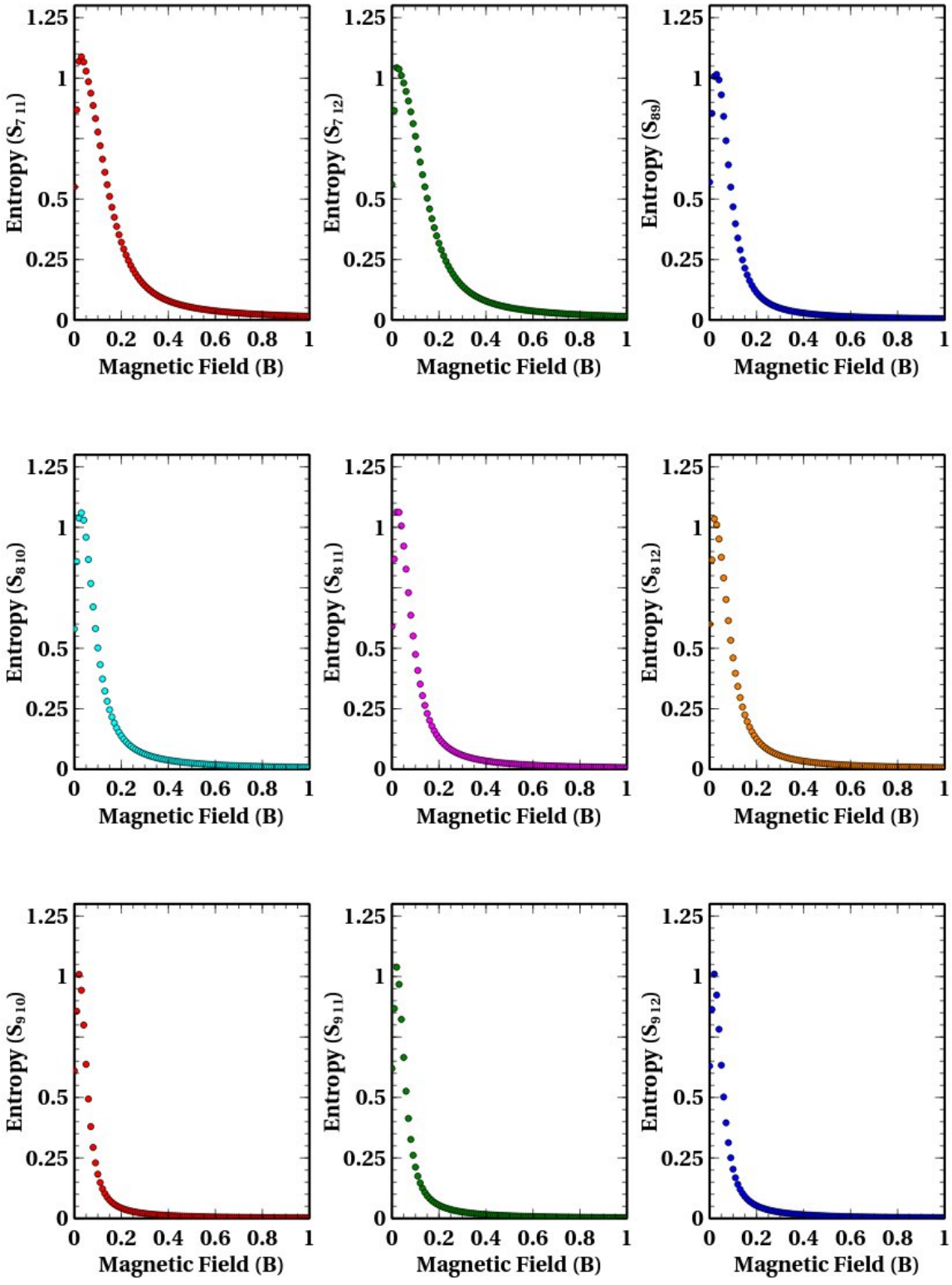

Entropy ($S_{3\,10}$)
Entropy ($S_{3\,11}$)
Entropy ($S_{3\,12}$)
Entropy (S_{45})
Entropy (S_{46})
Entropy (S_{47})
Entropy (S_{48})
Entropy (S_{49})
Entropy ($S_{4\,10}$)
Magnetic Field (B)
0
0.25
0.5
1
1.25
0.2
0.4
0.6
0.8

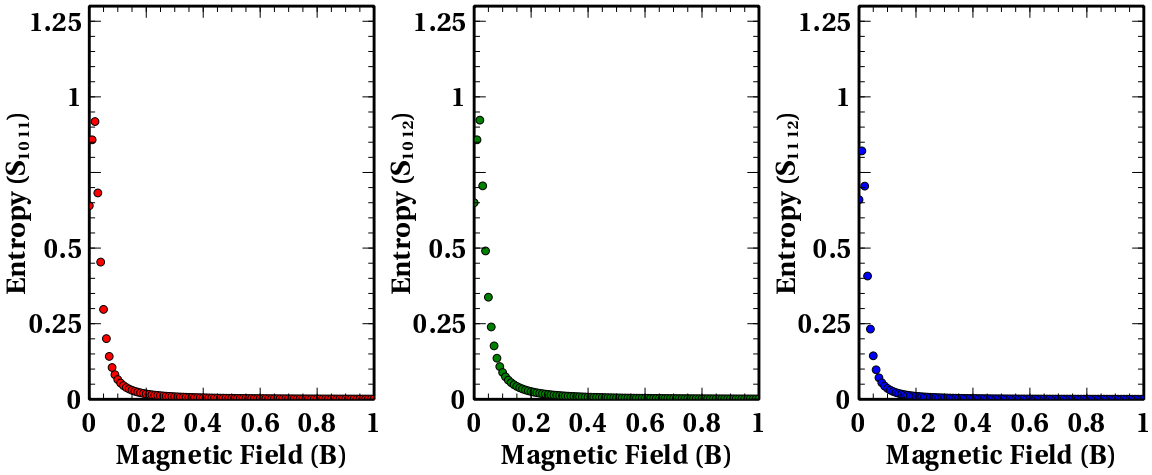

Entropy (S_{67})
Entropy (S_{68})
Entropy (S_{69})
Magnetic Field (B)
Magnetic Field (B)
Magnetic Field (B)


Entropy ($S_{6\,10}$)
Entropy ($S_{6\,11}$)
Entropy ($S_{6\,12}$)
Magnetic Field (B)
Magnetic Field (B)
Magnetic Field (B)


Entropy (S_{78})
Entropy (S_{79})
Entropy ($S_{7\,10}$)
Magnetic Field (B)
Magnetic Field (B)
Magnetic Field (B)

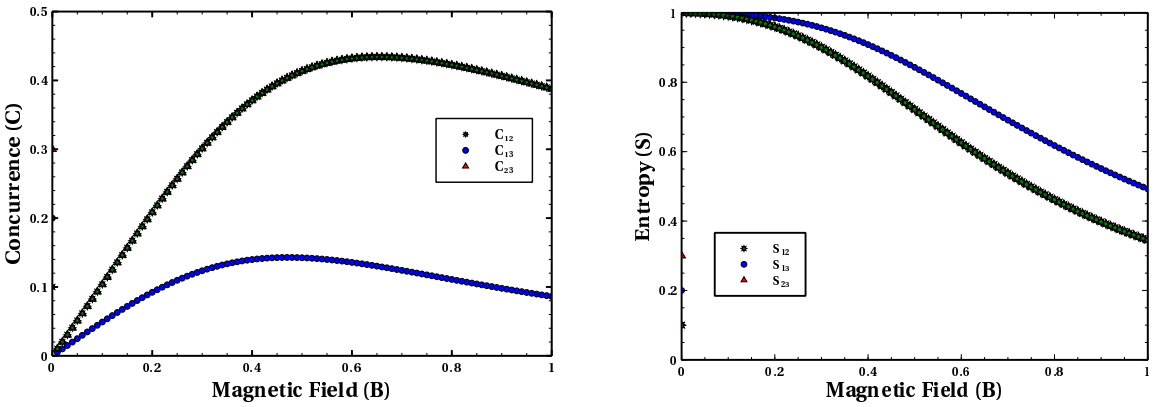

Entropy (S_{67})
Entropy (S_{68})
Entropy (S_{69})
Entropy (S_{610})
Entropy (S_{611})
Entropy (S_{612})
Entropy (S_{78})
Entropy (S_{79})
Entropy (S_{710})
Magnetic Field (B)
0
0.25
0.5
1
1.25
0.2
0.4
0.6
0.8


Entropy ($S_{7\,11}$)
Entropy ($S_{7\,12}$)
Entropy (S_{89})
Entropy ($S_{8\,10}$)
Entropy ($S_{8\,11}$)
Entropy ($S_{8\,12}$)
Entropy ($S_{9\,10}$)
Entropy ($S_{9\,11}$)
Entropy ($S_{9\,12}$)
Magnetic Field (B)
0
0.25
0.5
1
1.25
0.2
0.4
0.6
0.8



4.3.3 Monogamy Of Entanglement(MOE)

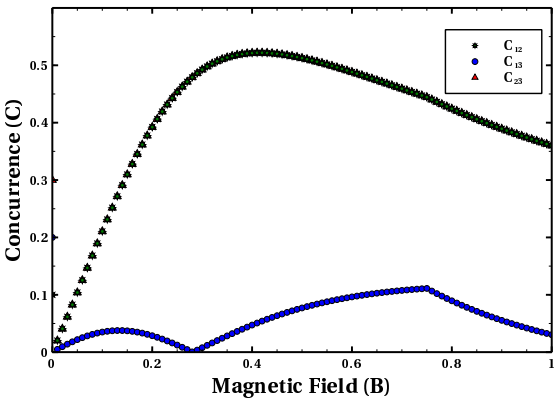
As we have already discussed for both open and periodic chain systems, C_{12}, C_{23}, and C_{13} are increasing with the increase in the applied magnetic field which signifies that the entanglement between these qubit pairs increases with the increase in magnetic field and the decrease of S_{12}, S_{23} and S_{13} with the increase in magnetic field signifies that the amount of entanglement of the blocks (consisting these qubit pairs) with the other blocks(corresponding to the rest of the qubits respectively) decreases with the increase in the field. This nature of concurrence and block entropy variation indicates the MOE nature. It is remarkable to see that the MOE is observed even in such random spin chain systems.



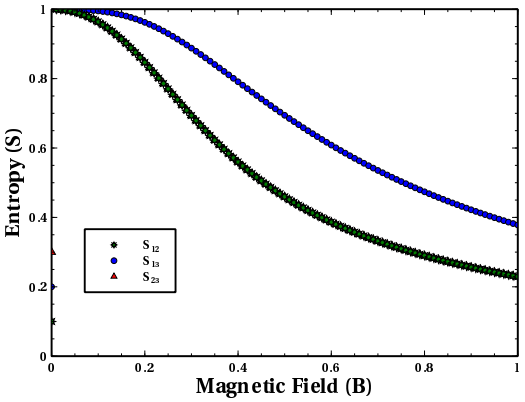
(a) Concurrence for the 3-qubit Ising-like open chain system

(b) Block entropy for the 3-qubit Ising-like open chain system

Figure 4.11: Study of the monogamy of entanglement for a three-qubit open-chain Ising-like system with inverse Fibonacci interaction.



(a) Concurrence Study for the 3-qubit Ising-like periodic chain system



(b) Block entropy Study for 3-qubit periodic chain Ising-like system

Figure 4.12: Study of the monogamy of entanglement for three-qubit periodic chain Ising-like systems with inverse Fibonacci interaction.

4.4 Chapter Summary

At the beginning of the chapter, we discussed the idea of quantum entanglement and its importance. We discussed concurrence and block entropy in the context of quantum entanglement and the monogamy of entanglement (MOE). Then we took the Fibonacci Ising system where the interaction between the nearest neighboring qubits was varied as elements of the inverse Fibonacci series for the entanglement study. We had studied three-qubit and twelve-qubit systems in both open chain as well as periodic chain conditions. We had observed that the nearest neighbour qubit pairs are more entangled compared to the other qubit pairs. We also observed that the concurrences corresponding to the nearest neighbour qubit pairs increase with the application of the external magnetic field, reach a maximum value, and then decrease to some extent. For the block entropy study, we saw that the block consisting of the nearest neighbour qubit pairs is less entangled with the other block containing the remaining number of qubits. It indicated the monogamy of entanglement even in such type of random Ising-like spin chain system.

★

Chapter 5

Entanglement Study in Heisenberg-type Spin Chain systems

5.1 Introduction

In this chapter, we are aiming to study the eigenvalue spectrum and quantum entanglement between the qubits when considering the Heisenberg-like spin chain systems involving inverse Fibonacci-type interaction strengths between the qubit pairs. In Chapter 1, we had already discussed the structure of the Hamiltonian and also saw how it can be derived.

5.2 Numerical Solution of Heisenberg-like Spin Chain System Consisting of N Number of qubits

We are interested in solving the Heisenberg-like spin chain system consisting of N number of qubits where the interaction energies corresponding to successive qubit pairs are taken as inverse Fibonacci sequence i.e. the interaction energy corresponding

to qubit pair 1 & 2 is $J_1 = \frac{1}{F_1}$ (as per our notation) that for qubit pair 2 & 3 is $J_2 = \frac{1}{F_2}$ and so on. We also take the external magnetic field along the X direction and vary it from 0 to 1 in steps of 0.01 units. So, the Hamiltonian that we are going to solve is given by,

$$\mathcal{H} = \sum_{i=1}^{N} J_i \vec{\sigma_i}.\vec{\sigma_{i+1}} + B \sum_{i=1}^{N} \sigma_i^x \tag{5.1}$$

If we simplify the Hamiltonian, it will look like this,

$$\mathcal{H} = \sum_{i=1}^{N} J_i(\sigma_i^x \sigma_{i+1}^x + \sigma_i^y \sigma_{i+1}^y + \sigma_i^z \sigma_{i+1}^z) + B \sum_{i=1}^{N} \sigma_i^x \tag{5.2}$$

where,

$$\sigma^x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \sigma^y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \sigma^z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{5.3}$$

are the X, Y, and Z components of the Pauli spin matrix. The Hamiltonian matrix can be elegantly constructed using the tensor product approach as previously done in the context of the Ising model in Chapter 1, but as discussed earlier, for a higher number of qubits, it will require a large amount of computational time as well as memory allocation. So we have shifted to our alternative approach, which involves the process of bit test. One can easily check that,

$$\sigma^x|0\rangle = |1\rangle, \qquad \sigma^x|1\rangle = |0\rangle,$$
$$\sigma^y|0\rangle = i|1\rangle = (-1)^0\ i|1\rangle, \qquad \sigma^y|1\rangle = -i|0\rangle = (-1)^1\ i|0\rangle,$$
$$\sigma^z|0\rangle = |0\rangle, \qquad \sigma^z|1\rangle = -|1\rangle.$$

So if we consider a three-qubit system, then in that case,

$$\sigma_1^x \sigma_2^x|000\rangle = \sigma_1^x|010\rangle = |110\rangle,$$
$$\sigma_1^x \sigma_2^x|111\rangle = \sigma_1^x|101\rangle = |001\rangle.$$

So we can generalize the operation as follows,

$$\sigma_1^x \sigma_2^x|a_1 a_2 a_3\rangle = |1 - a_1 \quad 1 - a_2 \quad a_3\rangle. \tag{5.4}$$

Again,

$$\sigma_1^y\sigma_2^y|000\rangle = i\sigma_1^y|010\rangle = -|110\rangle,$$
$$\sigma_1^y\sigma_2^y|111\rangle = (-1).i\sigma_1^y|101\rangle = (-1)(-1)(-1)|001\rangle = -|001\rangle.$$

So we can generalize the operation as follows,

$$\sigma_1^y\sigma_2^y|a_1a_2a_3\rangle = (-1)^{a_1+a_2+1}|1-a_1 \quad 1-a_2 \quad a_3\rangle. \tag{5.5}$$

Similarly, we can check,

$$\sigma_1^z\sigma_2^z|000\rangle = \sigma_1^z|000\rangle = |000\rangle,$$
$$\sigma_1^z\sigma_2^z|111\rangle = (-1)\sigma_1^z|111\rangle = (-1)(-1)|111\rangle = |111\rangle.$$

So we can generalize the operation as follows,

$$\sigma_1^z\sigma_2^z|a_1a_2a_3\rangle = (1-2a_1)(1-2a_2)|a_1a_2a_3\rangle \tag{5.6}$$

Now we are in a position to generalize the operations of $\sigma_i^x\sigma_{i+1}^x$, $\sigma_i^y\sigma_{i+1}^y$ and $\sigma_i^z\sigma_{i+1}^z$ on i^{th} and $(i+1)^{th}$ qubit for system comprising 'N' number of qubits as follows,

$$\sigma_i^x\sigma_{i+1}^x|a_1a_2a_3...a_ia_{i+1}...a_N\rangle = |a_1a_2...1-a_i\ 1-a_{i+1}...a_N\rangle \tag{5.7}$$
$$\sigma_i^y\sigma_{i+1}^y|a_1a_2a_3...a_ia_{i+1}...a_N\rangle = (-1)^{a_i+a_{i+1}+1}|a_1a_2...1-a_i\ 1-a_{i+1}...a_N\rangle \tag{5.8}$$
$$\sigma_1^z\sigma_2^z|a_1a_2a_3...a_ia_{i+1}...a_N\rangle = (1-2a_i)(1-2a_{i+1})|a_1a_2a_3...a_ia_{i+1}...a_N\rangle. \tag{5.9}$$

Utilizing the above, we can construct the Fortran code to build the Hamiltonian of the Heisenberg-Fibonacci system.

5.2.1 Case-(1):Open Chain System

This code is made to study the eigenvalue spectrum (i.e., the variation of energy eigenvalue to the magnetic field) corresponding to the N-qubit open chain system of the Heisenberg-Fibonacci spin chain model.

```
program Heisenberg_model_with_fibonacci
!For n qubit system
!when J is varied like the inverse of the Fibonacci series and B is
   also varied from 0 to 1 in steps of 0.01.
!Open chain System
implicit none
integer :: i, n, num, o, a, x, y, k,  differing_bits
integer, parameter :: q = 1, t = 1, c = 1
real*8:: B, dB
integer:: m
integer, parameter:: d = 2
character(len=:), allocatable :: state(:)
integer, allocatable:: decimal(:)
integer, allocatable:: psi(:)
double precision, allocatable :: p(:), vec1(:), res1(:), vit(:),
   rho(:,:)
integer:: j
real*8, allocatable :: S1(:,:), S2(:,:),S3(:,:),S4(:,:),S5(:,:), H
   (:,:), H1(:,:)
integer, allocatable:: site(:)

! Parameters for LAPACK DSYEV
integer:: lda, info, lwork
real*8, allocatable :: w(:), work(:), dw(:)
character*1:: JOBZ, UPLO

! Input values
print*, "Enter the number of bits you want:"
read*, n
num = 2**n ! Calculate number of states
m = n

! Allocate arrays after determining the value of num
allocate(character(len=n) :: state(num))
allocate(integer :: decimal(num))
allocate(psi(n))
allocate(S1(num, num))
```

```
allocate(S2(num, num))
allocate(S3(num, num))
allocate(S4(num, num))
allocate(S5(num, num))
allocate(H(num, num))
allocate(H1(num, num))
allocate(vec1(num))
allocate(vit(0:num-1))
allocate(p(m))
allocate(rho(num,num))
allocate(site(0:d-1))

! LAPACK parameters
lda = num
lwork = 3 * num - 1
allocate(real(8) :: w(lda), work(work))

! Printing the possible states
!print*, "The states are:"
do i = 0, num - 1
    call integer_binary(i, state(i + 1), n)  ! Adjusting the index
   for 1-based
    !print *, state(i + 1)
end do

! Convert binary states to decimal
do i = 1, num
    decimal(i) = btod(state(i), n)
    !write(722,*) decimal(i)
end do

call FibonacciSeries(m, p)

!Perform calculation based on sigma_i^x*sigma_i+1^x as S1
! Initialize the S1 matrix
S1 = 0.0
do i = 1, num
    do j = 1, num
```

```
72          ! Initialize differing_bits counter
73          differing_bits = 0
74          do k = 0, n-1
75          ! Compare the k-th bit of the i-th and j-th states
76          if (btest(decimal(i), k) .neqv. btest(decimal(j), k)) then
77              differing_bits = differing_bits + 1
78          end if
79          end do
80          ! Check if exactly two bits differ, and those bits are the
    specified positioned two qubits
81          if (differing_bits == 2) then
82              do o = 1,(n-1),1
83              !print*, i, j, decimal(i), decimal(j)
84              if ((btest(decimal(i), (n-o)) .neqv. btest(decimal(j), (
    n-o))) .and. &
85              (btest(decimal(i), (n-o-1)) .neqv. btest(decimal(j), (n-
    o-1)))) then
86              S1(i, j) = p(o)
87              !print*, i, j, o, btest(decimal(i), (n-o)), btest(
    decimal(j), (n-o)), p(o), S1(i,j)
88              end if
89              end do
90          else
91          S1(i, j) = 0.0
92          end if
93      end do
94  end do
95
96  !Perform calculation based on sigma_i^y*sigma_i+1^y as S2
97  ! Initialize the S2 matrix
98  S2 = 0.0
99  do i = 1, num
100     do j = 1, num
101     call single_digit(state(i), psi)
102     ! Initialize differing_bits counter
103     differing_bits = 0
104         do k = 0, n-1
105         ! Compare the k-th bit of the i-th and j-th states
```

```
106         if (btest(decimal(i), k) .neqv. btest(decimal(j), k)) then
107             differing_bits = differing_bits + 1
108         end if
109         end do
110         ! Check if exactly two bits differ, and those bits are the
    specified positioned two qubits
111         if (differing_bits == 2) then
112             do o = 1,(n-1),1
113                 if ((btest(decimal(i), (n-o)) .neqv. btest(decimal(j
    ), (n-o))) .and. &
114                 (btest(decimal(i), (n-o-1)) .neqv. btest(decimal(j),
     (n-o-1)))) then
115                 S2(i, j) = (-1)**(psi(o) + psi(o+1) + 1)*p(o)
116                 end if
117             end do
118         else
119         S2(i, j) = 0.0
120         end if
121     end do
122 end do
123
124 !Perform calculation based on sigma_i^z*sigma_i+1^z$ as S3
125 ! Perform calculations based on the decimal values for S3
126 do i = 1, num
127     do j = 1, num
128     call single_digit(state(i), psi)
129     if (decimal(i) /= decimal(j)) then
130         S3(i, j) = 0.0
131     else
132         S3(i, j) = 0.0
133         do k = 2, n
134         S1(i, j) = S1(i, j) + p(k-1)*(1.0 - 2.0 * psi(k-1)) * (1.0 -
     2.0 * psi(k))
135         end do
136     end if
137     end do
138 end do
139
```

```
140 ! Construct the S4 matrix (for sigma_x )
141 S4 = 0.0
142 do i = 1, num
143     do j = 1, num
144     ! Initialize differing_bits counter
145     differing_bits = 0
146     do k = 1, n
147         ! Compare the k-th bit of the i-th and j-th states
148         if (btest(decimal(i), k-1) .neqv. btest(decimal(j), k-1))
    then
149         differing_bits = differing_bits + 1
150         end if
151     end do
152     ! If exactly one bit differs, set S4(i, j) = 1.0
153     if (differing_bits == 1) then
154         S4(i, j) = 1.0
155     else
156         S4(i, j) = 0.0
157     end if
158     end do
159 end do
160
161 ! Perform calculations based on the decimal values for S5
162 do i = 1, num
163     do j = 1, num
164     call single_digit(state(i), psi)
165     if (decimal(i) /= decimal(j)) then
166         S5(i, j) = 0.0
167     else
168         S5(i, j) = 0.0
169     do k = 1, n
170         S5(i, j) = S5(i,j) + (1.0 - 2.0 * psi(k))
171     end do
172     end if
173     end do
174 end do
175
176 ! Total Hamiltonian
```

```
177 do y = 0, 100, 1
178     B = y * 0.01
179     H =  (q*S1 + t*S2 + c*S3 ) + B * S4
180     ! Eigenvalue and eigenvector determination using DSYEV
181     JOBZ = 'V'
182     UPLO = 'U'
183     call DSYEV(JOBZ, UPLO, num, H, lda, w, work, lwork,info)
184     write(9,*) "Eigenvalue:"
185     do i = 1, lda
186         write(90,*) B, w(i)
187     end do
188 end do
189
190 deallocate(state)
191 deallocate(decimal)
192 deallocate(psi)
193 deallocate(S1)
194 deallocate(S2)
195 deallocate(S3)
196 deallocate(S4)
197 deallocate(S5)
198 deallocate(H)
199 deallocate(H1)
200 deallocate(p)
201 deallocate(rho)
202
203 contains
204
205 function btod(state, n) result(decimal)
206 implicit none
207 integer, intent(in) :: n
208 character(len=n) :: state
209 integer :: decimal, i
210 decimal = 0
211 do i = 1, n
212     if (state(i:i) == '1') then
213         decimal = decimal + 2**(n-i)
214     end if
```

```
end do
end function btod

subroutine integer_binary(num, binary_string, n)
    implicit none
    integer :: num
    integer, intent(in) :: n
    integer :: i, temp_num
    character(len=n), intent(out) :: binary_string
    temp_num = num
    binary_string = repeat('0', n)
    do i = 0, n - 1
    if (mod(temp_num, 2) == 1) then
        binary_string(n-i:n-i) = '1'
    else
    binary_string(n-i:n-i) = '0'
    end if
    temp_num = temp_num / 2
    end do
end subroutine integer_binary

subroutine single_digit(A, psi)
implicit none
character(len=*), intent(in) :: A
integer, intent(out) :: psi(:)
integer :: i
do i = 1, len(A)
    if (A(i:i) == '0') then
        psi(i) = 0
    else if (A(i:i) == '1') then
    psi(i) = 1
    else
    print *, "Invalid input in single_digit subroutine"
    end if
end do
end subroutine single_digit

subroutine FibonacciSeries(m, reciprocal)
```

```
253 implicit none
254 integer :: i
255 integer :: m
256 double precision :: fib(m)
257 double precision :: reciprocal(m)
258 if (m <= 0) then
259     print*, "Number of terms must be greater than 0."
260     stop
261 end if
262 ! Initialize the first two terms
263 fib(1) = 1
264 if (n > 1) then
265     fib(2) = 1
266 end if
267 ! Generate the Fibonacci series
268 do i = 3, m
269     fib(i) = fib(i-1) + fib(i-2)
270 end do
271 ! Compute reciprocals
272  do i = 1, m
273     reciprocal(i) = 1.0d0 / real(fib(i))
274 end do
275 !print*, fib
276 end subroutine FibonacciSeries
277
278 end program Heisenberg_model_with_fibonacci
```

For a three-qubit open chain system, the eigenvalue spectrum corresponding to the change in the magnetic field is given as follows in Fig.[5.1]. In our Hamiltonian, as per Eq.[5.1], the magnetic field was assumed to be along the X-direction. If instead the magnetic field is considered along the Y-direction or Z-direction, the eigenvalues of the Hamiltonian will remain unchanged compared to the present case due to the isotropy of the system.

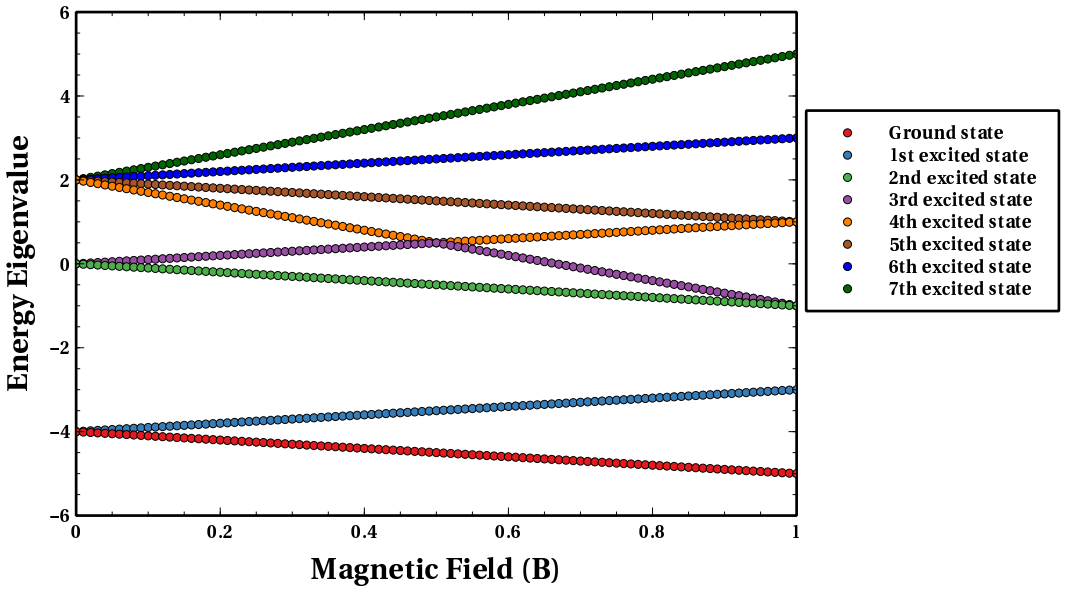


Figure 5.1: Energy eigenvalue spectrum with variation of magnetic field for Heisenberg-Fibonacci spin chain system consisting of 3 qubits

Even in this Heisenberg-Fibonacci spin chain, we can see clearly from the figure that the variation of the energy eigenvalues concerning the magnetic field is a straight line in nature. This is kind of obvious because the Hamiltonian of the system involves σ^2 and σ_x, which commute with each other as you can see below.

$$[\sigma^2, \sigma_x] = [\sigma_x^2 + \sigma_y^2 + \sigma_z^2, \sigma_x] = [\sigma_x^2, \sigma_x] + [\sigma_y^2, \sigma_x] + [\sigma_z^2, \sigma_x] \tag{5.10}$$

We know that,

$$[\sigma_x, \sigma_y] = i\sigma_z, \qquad [\sigma_y, \sigma_z] = i\sigma_x, \qquad [\sigma_z, \sigma_x] = i\sigma_y \tag{5.11}$$

Now,

$$[\sigma_x^2, \sigma_x] = 0 \tag{5.12}$$

and

$$\begin{aligned} [\sigma_y^2, \sigma_x] &= \sigma_y[\sigma_y, \sigma_x] + [\sigma_y, \sigma_x]\sigma_y, \\ &= \sigma_y(-i\sigma_z) + (-i\sigma_z)\sigma_y, \\ &= -i\sigma_y\sigma_z - i\sigma_z\sigma_y. \end{aligned} \tag{5.13}$$

similarly,

$$\begin{aligned}[\sigma_z^2, \sigma_x] &= \sigma_z[\sigma_z, \sigma_x] + [\sigma_z, \sigma_x]\sigma_z, \\ &= \sigma_z(i\sigma_y) + (i\sigma_y)\sigma_z, \\ &= i\sigma_z\sigma_y + i\sigma_y\sigma_z. \end{aligned} \tag{5.14}$$

Combining all these separate commutations, Eq.[5.10] boils down to,

$$[\sigma^2, \sigma_x] = [\sigma_x^2, \sigma_x] + [\sigma_y^2, \sigma_x] + [\sigma_z^2, \sigma_x] = 0 \tag{5.15}$$

If we consider $|\psi_g\rangle$ to be the ground state of the system, then we can get,

$$\langle\psi_g|\mathcal{H}|\psi_g\rangle = \langle\psi_g|\sum_i J_i\vec{\sigma}_i.\vec{\sigma}_{i+1}|\psi_g\rangle + B\langle\psi_g|\sum_i \sigma_i^x|\psi_g\rangle \tag{5.16}$$

which turns out to be,

$$E_g = a + Bu \tag{5.17}$$

Where E_g is the ground state energy, a and u are constants given by $\langle\psi_g|\sum_i J_i\vec{\sigma}_i.\vec{\sigma}_{i+1}|\psi_g\rangle$ and $\langle\psi_g|\sum_i \sigma_i^x|\psi_g\rangle$ respectively.

5.2.2 Case-(2):Periodic Chain System

Let us now analyze the N-qubit periodic chain system. For that, we can use the Fortran code mentioned below.

```
program Heisenberg_model_with_fibonacci
!For n-qubit Periodic chain system
!Program of Heisenberg-like spin chain system, when J is varied like
    the inverse of the Fibonacci series and B, is also varied from 0
    to 1 in steps of 0.01
implicit none
integer :: i, n, num, o, a, x, y, k,  differing_bits
integer, parameter :: q = 1, t = 1, c = 1
real*8 :: B
integer :: m
character(len=:), allocatable :: state(:)
```

```
integer, allocatable :: decimal(:)
integer, allocatable :: psi(:)
double precision, allocatable :: p(:)
integer :: j
real*8, allocatable :: S1(:,:), S2(:,:),S3(:,:),S4(:,:), H(:,:), H1
    (:,:), rho(:,:)

! Parameters for LAPACK DSYEV
integer :: lda, info, lwork
real*8, allocatable :: w(:), work(:)
character*1 :: JOBZ, UPLO

! Input values
print*, "Enter the number of bits you want:"
read*, n
num = 2**n ! Calculate number of states
m = n
! Allocate arrays after determining the value of num
allocate(character(len=n) :: state(num))
allocate(integer :: decimal(num))
allocate(psi(n))
allocate(S1(num, num))
allocate(S2(num, num))
allocate(S3(num, num))
allocate(S4(num, num))
allocate(H(num, num))
allocate(H1(num, num))
allocate(p(m))
allocate(rho(num,num))

! LAPACK parameters
lda = num
lwork = 3 * num - 1
allocate(real(8) :: w(lda), work(lwork))

! Printing the possible states
!print*, "The states are:"
do i = 0, num - 1
```

```
47      call integer_binary(i, state(i + 1), n)  ! Adjusting index for
    1-based
48  end do
49
50  ! Convert binary states to decimal
51  do i = 1, num
52      decimal(i) = btod(state(i), n)
53  end do
54
55  !Generate the inverse Fibonacci sequence
56  call FibonacciSeries(m, p)
57
58  !Perform calculation based on \sigma_i^x\sigma_{i+1}^x as S1
59  ! Initialize the S1 matrix
60  S1 = 0.0
61  do i = 1, num
62      do j = 1, num
63      ! Initialize differing_bits counter
64      differing_bits = 0
65      do k = 0, n-1
66      ! Compare the k-th bit of the i-th and j-th states
67      if (btest(decimal(i), k) .neqv. btest(decimal(j), k)) then
68      differing_bits = differing_bits + 1
69      end if
70      end do
71      ! Check if exactly two bits differ, and those bits are the
    specified positioned two qubits
72      if (differing_bits == 2) then
73      do o = 1,(n-1),1
74          if ((btest(decimal(i), (n-o)) .neqv. btest(decimal(j), (n-o)
    )) .and. &
75          (btest(decimal(i), (n-o-1)) .neqv. btest(decimal(j), (n-o-1)
    ))) then
76              S1(i, j) = p(o)
77          end if
78      end do
79      if ((btest(decimal(i), (n-1)) .neqv. btest(decimal(j), (n-1))) .
    and. &
```

```
80      (btest(decimal(i), 0) .neqv. btest(decimal(j), 0))) then
81      S1(i, j) = p(n)
82      end if
83      else
84      S1(i, j) = 0.0
85      end if
86      end do
87  end do
88
89  !Perform calculation based on \sigma_i^y\sigma_{i+1}^y as S2
90  ! Initialize the S2 matrix
91  S2 = 0.0
92  do i = 1, num
93      do j = 1, num
94       call single_digit(state(i), psi)
95      ! Initialize differing_bits counter
96      differing_bits = 0
97      do k = 0, n-1
98      ! Compare the k-th bit of the i-th and j-th states
99      if (btest(decimal(i), k) .neqv. btest(decimal(j), k)) then
100     differing_bits = differing_bits + 1
101     end if
102     end do
103     ! Check if exactly two bits differ, and those bits are the
    specified positioned two qubits
104     if (differing_bits == 2) then
105     do o = 1,(n-1),1
106         if ((btest(decimal(i), (n-o)) .neqv. btest(decimal(j), (n-o)
    )) .and. &
107         (btest(decimal(i), (n-o-1)) .neqv. btest(decimal(j), (n-o-1)
    ))) then
108         S2(i, j) = (-1)**(psi(o) + psi(o+1) + 1)*p(o)
109         end if
110     end do
111     if ((btest(decimal(i), (n-1)) .neqv. btest(decimal(j), (n-1))) .
    and. &
112         (btest(decimal(i), 0) .neqv. btest(decimal(j), 0))) then
113     S2(i, j) = (-1)**(psi(n-2) + psi(n) + 1)*p(n)
```

```
    end if
    else
        S2(i, j) = 0.0
    end if
    end do
end do

!Perform calculation based on \sigma_i^z\sigma_{i+1}^z as S3
! Perform calculations based on the decimal values for S3
do i = 1, num
    do j = 1, num
    call single_digit(state(i), psi)
    if (decimal(i) /= decimal(j)) then
    S3(i, j) = 0.0
    else
    S3(i, j) = 0.0
    do k = 1, n - 1
        S3(i, j) = S3(i, j) + p(k)*(1.0 - 2.0 * psi(k)) * (1.0 - 2.0
    * psi(k + 1))
    end do
        S3(i, j) = S3(i, j) + p(n)*(1.0 - 2.0 * psi(n)) * (1.0 - 2.0
    * psi(1))
    end if
    end do
end do

! Construct the S4 matrix (for sigma_x )
S4 = 0.0
do i = 1, num
    do j = 1, num
    ! Initialize differing_bits counter
    differing_bits = 0
    do k = 1, n
    ! Compare the k-th bit of the i-th and j-th states
    if (btest(decimal(i), k-1) .neqv. btest(decimal(j), k-1)) then
        differing_bits = differing_bits + 1
    end if
    end do
```

```
150     !If exactly one bit differs, set S4(i, j) = 1.0
151     if (differing_bits == 1) then
152         S4(i, j) = 1.0
153     else
154         S4(i, j) = 0.0
155     end if
156     end do
157 end do
158 
159 ! Total Hamiltonian
160 do y = 0, 100, 1
161 B = y * 0.01
162 H =  (q*S1 + t*S2 + c*S3 ) + B * S4
163 ! Eigenvalue and eigenvector determination using DSYEV
164 JOBZ = 'V'
165 UPLO = 'U'
166 call DSYEV(JOBZ, UPLO, num, H, lda, w, work, lwork, info)
167 if (info == 0) then
168     do a = 1, lda
169         write(15,*) B, w(a) !plot for magfield vs eigenvalue
170     end do
171 else
172     write(15,*) 'Error in DSYEV, INFO=', info
173 end if
174 end do
175 
176 deallocate(state)
177 deallocate(decimal)
178 deallocate(psi)
179 deallocate(S1)
180 deallocate(S2)
181 deallocate(S3)
182 deallocate(H)
183 deallocate(H1)
184 deallocate(p)
185 deallocate(rho)
186 
187 contains
```

```
188
189 function btod(state, n) result(decimal)
190 implicit none
191 integer, intent(in) :: n
192 character(len=n) :: state
193 integer :: decimal, i
194 decimal = 0
195 do i = 1, n
196 if (state(i:i) == '1') then
197     decimal = decimal + 2**(n-i)
198 end if
199 end do
200 end function btod
201
202 subroutine integer_binary(num, binary_string, n)
203 implicit none
204 integer :: num
205 integer, intent(in) :: n
206 integer :: i, temp_num
207 character(len=n), intent(out) :: binary_string
208 temp_num = num
209 binary_string = repeat('0', n)
210 do i = 0, n - 1
211     if (mod(temp_num, 2) == 1) then
212     binary_string(n-i:n-i) = '1'
213     else
214     binary_string(n-i:n-i) = '0'
215     end if
216     temp_num = temp_num / 2
217 end do
218 end subroutine integer_binary
219
220 subroutine single_digit(A, psi)
221 implicit none
222 character(len=*), intent(in) :: A
223 integer, intent(out) :: psi(:)
224 integer :: i
225 do i = 1, len(A)
```

```
226 if (A(i:i) == '0') then
227     psi(i) = 0
228     else if (A(i:i) == '1') then
229     psi(i) = 1
230 else
231     print *, "Invalid input in single_digit subroutine"
232 end if
233 end do
234 end subroutine single_digit
235
236 subroutine FibonacciSeries(m, reciprocal)
237 implicit none
238 integer :: i
239 integer :: m
240 double precision :: fib(m)
241 double precision :: reciprocal(m)
242 if (m <= 0) then
243     print*, "Number of terms must be greater than 0."
244     stop
245 end if
246 ! Initialize the first two terms
247 fib(1) = 1
248 if (n > 1) then
249     fib(2) = 1
250 end if
251 ! Generate the Fibonacci series
252     do i = 3, m
253         fib(i) = fib(i-1) + fib(i-2)
254     end do
255 ! Compute reciprocals
256     do i = 1, m
257         reciprocal(i) = 1.0d0 / real(fib(i))
258     end do
259 end subroutine FibonacciSeries
260 end program Heisenberg_model_with_fibonacci
```

Now, we can choose $n = 3$ in the above program to study the 3-qubit periodic

spin chain system. Using this code, we can plot the eigenvalue variation with the magnetic field, and we can notice that this variation is also linear for the same reason as mentioned in the open chain system earlier.

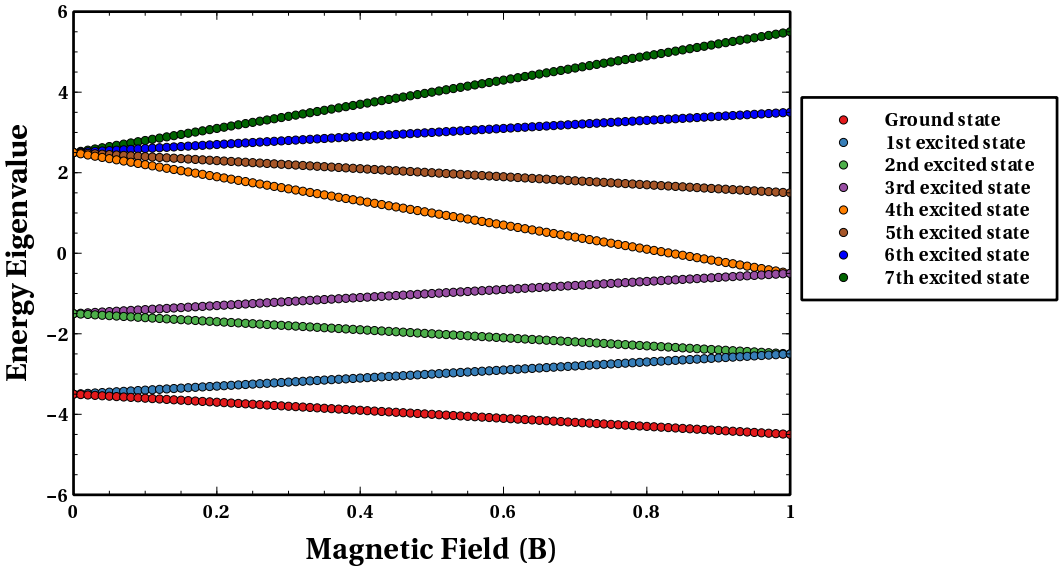


Figure 5.2: Energy eigenvalue spectrum with variation of magnetic field for Heisenberg-like periodic spin chain system consisting of 3 qubits

5.3 Entanglement Study in Heisenberg-Fibonacci Spin chain system

To study the entanglement between different qubit pairs, we have to consider the two tools - concurrence and von Neumann Entropy, which we had already discussed.

5.3.1 Studies on Concurrence

Let us first study the concurrences between different qubit pairs constituting the Heisenberg-Fibonacci spin chain system that we are discussing in this chapter. Again, we have to consider the two cases - case(1): Open chain System and case(2): Periodic Chain system. The concurrences corresponding to the ground state have been calculated in both cases.

5.3.1.1 Case(1): Open chain System

We have constructed the following Fortran code to study the concurrences between the qubit pairs for an N-qubit open chain system.

```
program Heisenberg_model_with_fibonacci_open_Concurrence
!For n qubit open chain system
!Program of generalised Ising Model when J is varied like inverse of
    Fibonacci series and B is also varied
implicit none
integer :: i, n, num, o, a, x, y, k,  differing_bits
integer, parameter :: q = 1, t = 1, c = 1
integer, parameter :: d = 3
real*8 :: B
real :: con, be2, con1
real*8 ::  be1
integer :: l
character(len=:), allocatable :: state(:)
integer, allocatable :: decimal(:)
integer, allocatable :: psi(:)
double precision, allocatable :: p(:), vec1(:), res1(:), vit(:)
integer :: j
real*8, allocatable :: S1(:,:), S2(:,:),S3(:,:),S4(:,:), S5(:,:), H
   (:,:), H1(:,:), rho(:,:)
integer, allocatable :: site(:)

! Parameters for LAPACK DSYEVX
integer :: LDA, info, LWORK, LDZ, M
real*8, allocatable :: Z(:,:)
integer, parameter :: IL = 1 , IU = 2
real*8, allocatable :: w(:), work(:)
character*1 :: JOBZ, UPLO
double precision :: VL, VU
integer, allocatable :: IFAIL(:), IWORK(:)
double precision, parameter :: ABSTOL = 0.0d0
character*1 :: RANGE   !'A' for computing all eigenvalues and 'I'
   for IL-th through IU-th eigenvalue

```

```
! Input values
print*, "Enter the number of bits you want:"
read*, n
num = 2**n ! Calculate number of states
l = n
! Allocate arrays after determining the value of num
allocate(character(len=n) :: state(num))
allocate(integer :: decimal(num))
allocate(psi(n))
allocate(S1(num, num))
allocate(S2(num, num))
allocate(S3(num, num))
allocate(S4(num, num))
allocate(S5(num, num))
allocate(H(num, num))
allocate(H1(num, num))
allocate(p(m))
allocate(vec1(num))
allocate(vit(0:num-1))
allocate(res1(num))
allocate(rho(num,num))
allocate(site(0:d-1))

! LAPACK parameters
LDA = num
LWORK = 8 * num + 1
M = IU-IL+1
LDZ = num
allocate(real(8) ::  w(LDA), work(LWORK))
allocate(integer :: IWORK(5*num), IFAIL(num))
allocate(real(8) :: Z(num,num))

! Printing the possible states
print*, "The states are:"
do i = 0, num - 1
    call integer_binary(i, state(i + 1), n)  ! Adjusting index for
   1-based
        print *, state(i + 1)
```

```
68 end do
69
70
71 ! Convert binary states to decimal
72     do i = 1, num
73         decimal(i) = btod(state(i), n)
74         !write(722,*) decimal(i)
75     end do
76
77 call FibonacciSeries(l, p)
78
79  !Perform calculation based on sigma_i^x*sigma_i+1^x as S1
80 ! Initialize the S1 matrix
81         S1 = 0.0
82         do i = 1, num
83             do j = 1, num
84                 ! Initialize differing_bits counter
85                 differing_bits = 0
86                 do k = 0, n-1
87                     ! Compare the k-th bit of the i-th and j-th
    states
88                     if (btest(decimal(i), k) .neqv. btest(decimal(j)
    , k)) then
89                         differing_bits = differing_bits + 1
90                     end if
91                 end do
92                 ! Check if exactly two bits differ, and those bits
    are the specified positioned two qubits
93                 if (differing_bits == 2) then
94                     do o = 1,(n-1),1
95                         !print*, i, j, decimal(i), decimal(j)
96                         if ((btest(decimal(i), (n-o)) .neqv. btest(
    decimal(j), (n-o))) .and. &
97                             (btest(decimal(i), (n-o-1)) .neqv. btest
    (decimal(j), (n-o-1)))) then
98                             S1(i, j) = p(o)
99                             !print*, i, j, o, btest(decimal(i), (n-o
    )), btest(decimal(j), (n-o)), p(o), S1(i,j)
```

```
                        end if
                    end do
                else
                    S1(i, j) = 0.0
                end if

            end do
        end do

!Perform calculation based on sigma_i^y*sigma_i+1^y as S2
! Initialize the S2 matrix
S2 = 0.0
do i = 1, num
    do j = 1, num
    call single_digit(state(i), psi)
    ! Initialize differing_bits counter
    differing_bits = 0
    do k = 0, n-1
    ! Compare the k-th bit of the i-th and j-th states
        if (btest(decimal(i), k) .neqv. btest(decimal(j), k)) then
        differing_bits = differing_bits + 1
        end if
    end do
    ! Check if exactly two bits differ, and those bits are the specified positioned two qubits
    if (differing_bits == 2) then
        do o = 1,(n-1),1
        !print*, i, j, decimal(i), decimal(j)
            if ((btest(decimal(i), (n-o)) .neqv. btest(decimal(j), (n-o))) .and. &
            (btest(decimal(i), (n-o-1)) .neqv. btest(decimal(j), (n-o-1)))) then
            S2(i, j) = (-1)**(psi(o) + psi(o+1) + 1)*p(o)
            !print*, i, j, o, btest(decimal(i), (n-o)), btest(decimal(j), (n-o)), p(o), S1(i,j)
            end if
        end do
    else
```

```
134         S2(i, j) = 0.0
135     end if
136     end do
137 end do
138 
139 !Perform calculation based on sigma_i^z*sigma_i+1^z as S3
140 ! Perform calculations based on the decimal values for S3
141 do i = 1, num
142     do j = 1, num
143         call single_digit(state(i), psi)
144         if (decimal(i) /= decimal(j)) then
145             S3(i, j) = 0.0
146         else
147         S3(i, j) = 0.0
148         do k = 2, n
149             S3(i, j) = S3(i, j) + p(k-1)*(1.0 - 2.0 * psi(k-1)) *
    (1.0 - 2.0 * psi(k))
150         end do
151         end if
152     end do
153 end do
154 
155 ! Construct the S4 matrix When magnetic field is along x direction (
    for sigma_x )
156 S4 = 0.0
157 do i = 1, num
158     do j = 1, num
159     ! Initialize differing_bits counter
160         differing_bits = 0
161         do k = 1, n
162             ! Compare the k-th bit of the i-th and j-th states
163             if (btest(decimal(i), k-1) .neqv. btest(decimal(j), k-1)
    ) then
164                 differing_bits = differing_bits + 1
165             end if
166         end do
167         ! If exactly one bit differs, set S4(i, j) = 1.0
168         if (differing_bits == 1) then
```

```
169             S4(i, j) = 1.0
170         else
171             S4(i, j) = 0.0
172         end if
173     end do
174 end do
175
176 !constructing the matrix S5 When the magnetic field is along the z
    direction (for sigma_z )
177 do i = 1, num
178     do j = 1, num
179     call single_digit(state(i), psi)
180     if (decimal(i) /= decimal(j)) then
181         S5(i, j) = 0.0
182     else
183         S5(i, j) = 0.0
184     do k = 1, n
185         S5(i, j) = S5(i,j) + (1.0 - 2.0 * psi(k))
186     end do
187     end if
188     end do
189 end do
190
191 ! Total Hamiltonian
192 do y = 0, 100, 1
193     B = y * 0.01
194     H =  (q*S1 + t*S2 + c*S3 ) + B * S4
195     ! Ground State and 1st excited state Eigenvalue and eigenvector
    determination using DSYEVX
196     JOBZ = 'V'
197     UPLO = 'U'
198     RANGE = 'I'
199     call DSYEVX (JOBZ, RANGE, UPLO, num, H, LDA, VL, VU, IL, IU,
    ABSTOL, M, W, Z, LDZ, WORK, LWORK, IWORK, IFAIL, INFO)
200     vec1 = Z(:, 1)
201     do i = 1, num
202         vit(i-1) = vec1(i)
203     end do
```

```
204     write(2,*) vit
205     if (info == 0) then
206         do a = 1, LDA
207             write(17,*) B, w(a) !plot for magfield vs eigenvalue
208         end do
209     else
210         write(15,*) 'Error in DSYEV, INFO=', info
211     end if
212 
213     !Concurrence
214     do i = 0, n-1
215         do j = i+1, n-1
216             if( i /= (n-1)) then
217             call CONVR(n,i,j,vit,con1)
218             write(8001,*) B , con1  !for 5 qubit
219             !write(84,*)  B, con1   !for 3 qubit
220             else
221                 EXIT
222             end if
223         end do
224     end do
225 end do
226 
227 deallocate(state)
228 deallocate(decimal)
229 deallocate(psi)
230 deallocate(S1)
231 deallocate(S2)
232 deallocate(S3)
233 deallocate(S4)
234 deallocate(S5)
235 deallocate(H)
236 deallocate(H1)
237 deallocate(p)
238 deallocate(rho)
239 deallocate(vit)
240 deallocate(vec1)
241 deallocate(site)
```

```
242
243 contains
244 function btod(state, n) result(decimal)
245     implicit none
246     integer, intent(in) :: n
247     character(len=n) :: state
248     integer :: decimal, i
249     decimal = 0
250     do i = 1, n
251         if (state(i:i) == '1') then
252             decimal = decimal + 2**(n-i)
253         end if
254     end do
255 end function btod
256
257     subroutine integer_binary(num, binary_string, n)
258         implicit none
259         integer :: num
260         integer, intent(in) :: n
261         integer :: i, temp_num
262         character(len=n), intent(out) :: binary_string
263         temp_num = num
264         binary_string = repeat('0', n)
265         do i = 0, n - 1
266             if (mod(temp_num, 2) == 1) then
267                 binary_string(n-i:n-i) = '1'
268             else
269                 binary_string(n-i:n-i) = '0'
270             end if
271             temp_num = temp_num / 2
272         end do
273     end subroutine integer_binary
274
275     subroutine single_digit(A, psi)
276         implicit none
277         character(len=*), intent(in) :: A
278         integer, intent(out) :: psi(:)
279         integer :: i
```

```
        do i = 1, len(A)
            if (A(i:i) == '0') then
                psi(i) = 0
            else if (A(i:i) == '1') then
                psi(i) = 1
            else
                print *, "Invalid input in single_digit subroutine"
            end if
        end do
    end subroutine single_digit

    subroutine FibonacciSeries(m, reciprocal)
        implicit none
        integer :: i
        integer :: m
        double precision :: fib(m)
        double precision :: reciprocal(m)
        if (m <= 0) then
            print*, "Number of terms must be greater than 0."
            stop
        end if
        ! Initialize the first two terms
        fib(1) = 1
        if (n > 1) then
            fib(2) = 1
        end if
        ! Generate the Fibonacci series
        do i = 3, m
            fib(i) = fib(i-1) + fib(i-2)
        end do
        ! Compute reciprocals
        do i = 1, m
            reciprocal(i) = 1.0d0 / real(fib(i))
        end do
        !print*, fib
    end subroutine FibonacciSeries
    subroutine CONVR(s,i0,j0,vin,con)
        implicit none
```

```
318         integer::s,i0,j0 ,site(0:1),i
319         real*8::vin(0:2**s-1),evs
320         real*8,dimension(0:3,0:3)::rdm,MM1,MM2,MM3,MM4
321         INTEGER::INFO, LDA, LWORK, NN,LDVL,LDVR
322         parameter(NN=4,LDA=NN,LWORK=4*NN,LDVL=NN,LDVR=NN)
323         real*8::WORK(0:LWORK-1),AA2(0:LDA-1,0:NN-1)
324         real*8::VL(0:NN-1,0:NN-1),VR(0:NN-1,0:NN-1)
325         real*8::WR(0:NN-1),WI(0:NN-1)
326         real*8::ev(NN),c2,x11,const,sigmay(0:3,0:3)
327         real :: con
328         sigmay(0,:)=(/0.0,0.0,0.0,-1.0/)
329         sigmay(1,:)=(/0.0,0.0,1.0,0.0/)
330         sigmay(2,:)=(/0.0,1.0,0.0,0.0/)
331         sigmay(3,:)=(/-1.0,0.0,0.0,0.0/)
332         site(0)=i0
333         site(1)=j0
334         call PTRVR(s,2,site,vin,rdm)
335         write(3,*) rdm
336         MM1=matmul(rdm,sigmay)
337         MM2=matmul(rdm,sigmay)
338         MM3=matmul(MM1,MM2)
339         call dgeev('N', 'N', NN, MM3, LDA, WR, &
340             WI, VL, LDVL, VR, LDVR, WORK, LWORK, INFO)
341         do i=1,NN,1
342             if (WR(i-1) .lt. 0.0000000000001) then
343                 WR(i-1)=0.0d0
344             end if
345                 ev(i)=WR(i-1)
346         end do
347         do i=2,4
348             if(ev(i)>=ev(1))then
349                 evs=ev(i)
350                 ev(i)=ev(1)
351                 ev(1)=evs
352             end if
353         end do
354         do i=3,4
355             if(ev(i)>=ev(2))then
```

```
356                 evs=ev(i)
357                 ev(i)=ev(2)
358                 ev(2)=evs
359             end if
360         end do
361         do i=4,4
362             if(ev(i)>=ev(3))then
363                 evs=ev(i)
364                 ev(i)=ev(3)
365                 ev(3)=evs
366             end if
367         end do
368         c2=0
369         con=0
370         c2=dsqrt(ev(1))-dsqrt(ev(2))-dsqrt(ev(3))-dsqrt(ev(4))
371         ! print*, c2
372         if(c2>0)then
373             con=c2
374         else
375            con=0
376         end if
377      end subroutine
378     subroutine PTRVR(s, s1, site, vin, rdm)
379         implicit none
380         integer :: s, s1, s2
381         real*8 :: trace
382         real*8, dimension(0:2**s-1) :: vin
383         real*8, dimension(0:2**s1-1, 0:2**s1-1) :: rdm
384         integer :: ii, i1, a, i0, ia, j1, oo, k1, x1, y1, j, t1, t2,
    z1
385         integer, dimension(0:s1-1) :: site, site2
386         integer, dimension(0:s1-1) :: bin
387         integer, dimension(0:(2**(s-s1))*(2**s1)-1) :: ind
388         s2 = s - s1
389         x1 = 0
390         y1 = 0
391         ind = 0
392         do j1 = 2**s1-1, 0, -1
```

```
393             do ii = 0, 2**s-1
394                 a = ii
395                 do i1 = 0, s1-1
396                     i0 = site(i1)
397                     call DTOBONEBIT(a, ia, i0, s)
398                     site2(i1) = ia
399                 end do
400                 call DTOB(j1, bin, s1)
401                 oo = 1
402                 do k1 = 0, s1-1
403                     oo = oo * (bin(k1) - site2(k1))
404                 end do
405                 if (abs(oo) == 1) then
406                     ind(x1) = ii
407                     x1 = x1 + 1
408                 end if
409             end do
410         end do
411 
412             rdm = 0.0d0
413             do t1 = 0, 2**s1-1
414                 do t2 = 0, 2**s1-1
415                     do z1 = 0, 2**s2-1
416                         rdm(t1, t2) = rdm(t1, t2) + vin(ind((2**s2)
    * t1 + z1)) * vin(ind((2**s2) * t2 + z1))
417                     end do
418                 end do
419             end do
420         end subroutine
421 
422         subroutine DTOBONEBIT(m, ia, i0, s)
423             implicit none
424             integer*4 :: s
425             integer :: tt(0:s-1)
426             integer :: m, ia, i0
427             call DTOB(m, tt, s)
428             ia = tt(i0)
429         end subroutine
```

```
430
431         subroutine DTOB(m, tt, s)
432             implicit none
433             integer :: s
434             integer, dimension(0:s-1) :: tt
435             integer :: m, k, a2
436             tt = 0
437             a2 = m
438             do k = 0, s-1
439                 tt(s-k-1) = mod(a2, 2)
440                 a2 = a2 / 2
441                 if (a2 == 0) exit
442             end do
443         end subroutine
444
445 end program Heisenberg_model_with_fibonacci_open_concurrence
```

It is to be mentioned here that, as we already discussed that we are interested in calculating the concurrences corresponding to the ground state of the system, so we have used *DSYEVX* LAPACK subroutine instead of *DSYEV* for calculating the ground state only. Now, for the three-qubit system, we have to give the input $n = 3$; then, after compiling the above code, we shall be able to calculate the concurrences corresponding to qubit pairs (1,2), (1,3) & (2,3). The variations of these concurrences are shown in the following Fig.[5.3]. The Fig.[5.3] shows that the concurrences are unaffected by the change in the magnetic field, and the concurrence corresponding to the qubit pairs (1,2) and (2,3) is more than the concurrence corresponding to the qubit pair (1,3).

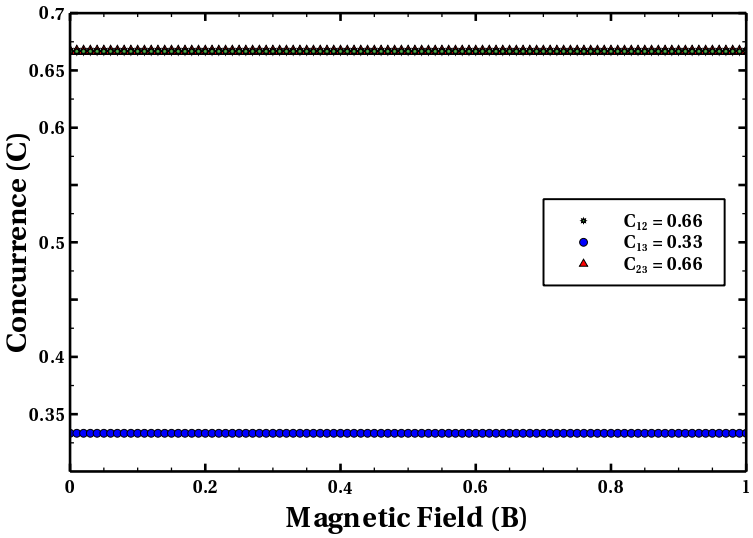


Figure 5.3: Variation of Concurrences with magnetic field for three-qubit open chain Heisenberg-like system.

Let us see what about the concurrences for 4 qubit open chain system. For that purpose, we can give the value of $n = 4$ in the above program. We can see from the Fig.[5.4] below shows how the concurrences corresponding to different qubit pairs (1,2), (1,3), (1,4),(2,3), (2,4) & (3,4) vary with the change in magnetic field.

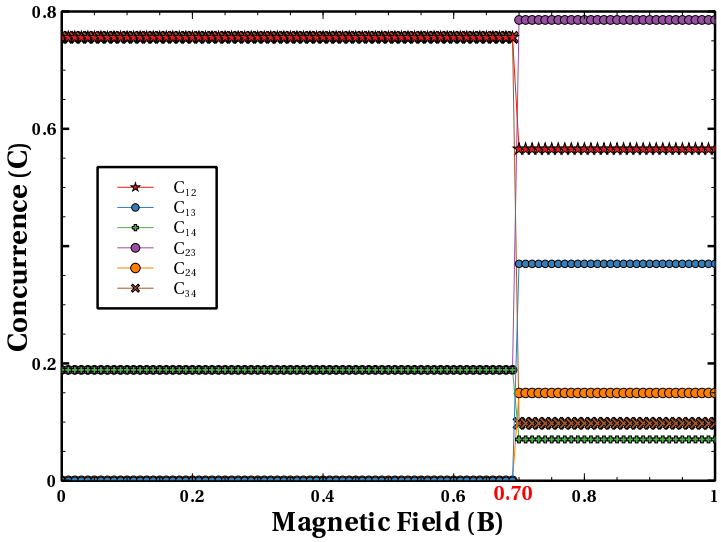


Figure 5.4: Variation of Concurrences with magnetic field for four-qubit open chain Heisenberg-like system.

The Fig.[5.4] shows that the concurrences have a discontinuity at the magnetic field of 0.70 units. Let's try to discuss the reason behind this.

5.3.1.2 Reason for the discontinuity in concurrences

We have calculated the concurrences from the ground state of the system. Now at the magnetic field of 0.70 units for the 4 qubit open chain system undergoes in quantum phase transition which results into the change of the ground state eigenvector i.e., at $B < 0.70$ unit if $|\psi_1\rangle$ be the ground state then at $B >$0.70 unit another eignstate $|\psi_2\rangle$ becomes the ground state. Since the eigenstates $|\psi_1\rangle$ & $|\psi_2\rangle$ have different entanglement properties, the concurrence and entropy calculated from them result in an abrupt change. We can take a look at the variation of the ground state eigenvalue with the change in the applied magnetic field.

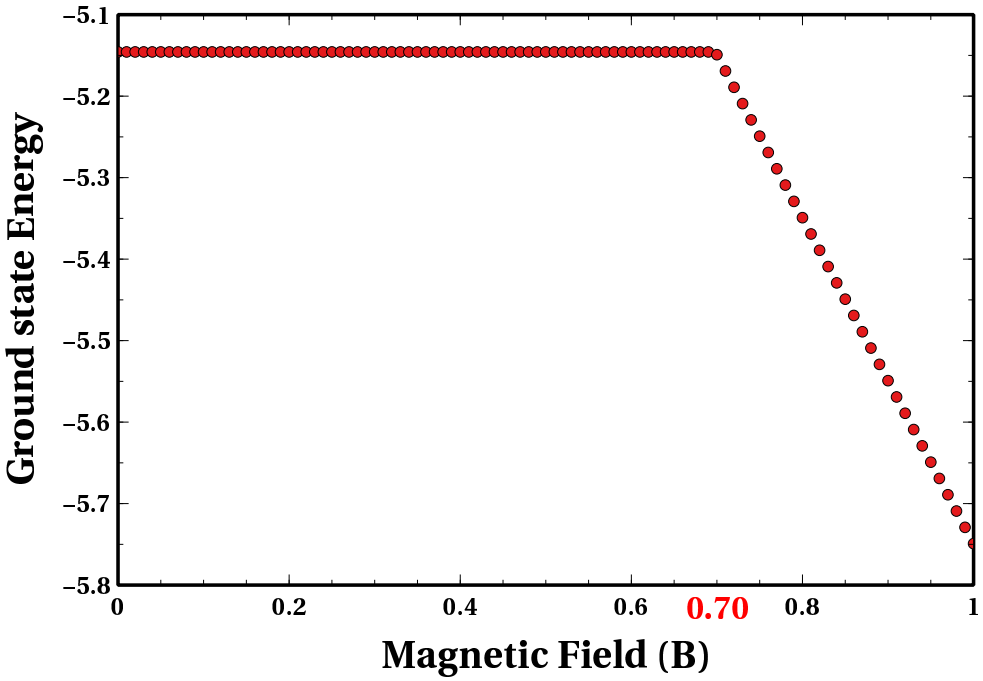


Figure 5.5: Variation of ground state energy with magnetic field for a 4-qubit open chain system corresponding to the Heisenberg-Fibonacci model.

From the Fig.[5.5], it is clear that the ground state eigenvalue variation with the magnetic field is also discontinuous at the magnetic field of 0.70 units of magnetic field. Now, we have observed that below 0.70 units of magnetic field for the 4-qubit

open chain system, the ground state eigenvector corresponding to the system has the structure,

$$|\psi_1\rangle = a(|0011\rangle + |1100\rangle) + b(|0101\rangle + |1010\rangle) + c(|0110\rangle + |1001\rangle) \tag{5.18}$$

and the corresponding density matrix has the form,

$$\rho = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & a^2 & 0 & ab & ac & 0 & 0 & ac & ab & 0 & a^2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & ab & 0 & b^2 & bc & 0 & 0 & bc & b^2 & 0 & ab & 0 & 0 & 0 \\
0 & 0 & 0 & ac & 0 & bc & c^2 & 0 & 0 & c^2 & bc & 0 & ac & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & ac & 0 & bc & c^2 & 0 & 0 & c^2 & bc & 0 & ac & 0 & 0 & 0 \\
0 & 0 & 0 & ab & 0 & b^2 & bc & 0 & 0 & bc & b^2 & 0 & ab & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & a^2 & 0 & ab & ac & 0 & 0 & ac & ab & 0 & a^2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix} \tag{5.19}$$

By partial tracing, the two-qubit reduced density matrices corresponding to this have the structure like,

$$\rho_{ij} = \begin{bmatrix}
a^2 & 0 & 0 & 0 \\
0 & b^2 + c^2 & 2bc & 0 \\
0 & 2bc & b^2 + c^2 & 0 \\
0 & 0 & 0 & a^2
\end{bmatrix}, \quad \text{where } i, j \in \{1, 2, 3, 4\}, \text{ provided } i < j. \tag{5.20}$$

We know that the trace of any density matrix is unity. So we can write that,

$$\begin{aligned} tr(\rho_{ij}) &= a^2 + (b^2 + c^2) + (b^2 + c^2) + a^2 \\ &= 2(a^2 + b^2 + c^2) \\ &= 1 \end{aligned} \tag{5.21}$$

$$\begin{aligned} 2(a^2 + b^2 + c^2) &= 1 \\ b^2 + c^2 &= \frac{1}{2} - a^2 \end{aligned} \tag{5.22}$$

Again, we can express these matrix elements of the two-qubit reduced density matrix ρ_{ij} in terms of the correlation functions as follows. From the definition of the correlation function, we can write,

$$\begin{aligned} G_{xx} &= \langle \sigma_{ij}^x \ \sigma_{ij}^x \rangle, \\ &= tr(\sigma_{ij}^x \ \sigma_{ij}^x \ \rho_{ij}), \\ &= 4bc. \end{aligned} \tag{5.23}$$

$$bc = \frac{1}{4} G_{xx}. \tag{5.24}$$

$$\begin{aligned} G_{zz} &= \langle \sigma_{ij}^z \ \sigma_{ij}^z \rangle, \\ &= tr(\sigma_{ij}^z \ \sigma_{ij}^z \ \rho_{ij}), \\ &= 2(a^2 - b^2 - c^2). \end{aligned} \tag{5.25}$$

Similarly,

$$\begin{aligned} G_{yy} &= \langle \sigma_{ij}^y \ \sigma_{ij}^y \rangle, \\ &= tr(\sigma_{ij}^y \ \sigma_{ij}^y \ \rho_{ij}), \\ &= 4bc. \end{aligned} \tag{5.26}$$

Combining Eq.[5.22] and Eq.[5.25],

$$4a^2 = 1 + G_{zz}, \tag{5.27}$$

$$a^2 = \frac{1}{4}(1 + G_{zz}). \tag{5.28}$$

Using Eq.[5.28], we can write that,

$$\begin{aligned} b^2 + c^2 &= \frac{1}{2} - a^2 \\ &= \frac{1}{2} - \frac{1}{4}(1 + G_{zz}) \end{aligned} \tag{5.29}$$

Now,

$$\begin{aligned}(b+c)^2 &= (b^2+c^2)+2bc,\\ &= (\frac{1}{2}-\frac{1}{4}(1+G_{zz}))+\frac{1}{4}(G_{xx}+G_{yy}),\\ &= \frac{1}{4}-\frac{1}{4}(G_{zz}-G_{xx}-G_{yy}).\end{aligned} \tag{5.30}$$

Again,

$$\begin{aligned}(b-c)^2 &= (b^2+c^2)-2bc,\\ &= (\frac{1}{2}-\frac{1}{4}(1+G_{zz}))-\frac{1}{4}(G_{xx}+G_{yy}),\\ &= \frac{1}{4}-\frac{1}{4}(G_{zz}+G_{xx}+G_{yy}).\end{aligned} \tag{5.31}$$

Now we can calculate the 2-qubit concurrence(C_{ij}) corresponding to the reduced density matrix ρ_{ij}. For that, we have to calculate the $\tilde{\rho}_{ij}$ which can be calculated as follows,

$$\begin{aligned}\tilde{\rho}_{ij} &= (\sigma^y\otimes\sigma^y)\rho^*_{ij}(\sigma^y\otimes\sigma^y),\\ &= (\sigma^y\otimes\sigma^y)\rho_{ij}(\sigma^y\otimes\sigma^y). \qquad \text{since all elements of } \rho_{ij} \text{ are real.}\end{aligned} \tag{5.32}$$

It boils down to the following,

$$\tilde{\rho}_{ij} = \begin{bmatrix} a^2 & 0 & 0 & 0\\ 0 & b^2+c^2 & 2bc & 0\\ 0 & 2bc & b^2+c^2 & 0\\ 0 & 0 & 0 & a^2\end{bmatrix} \tag{5.33}$$

Therefore,

$$\rho_{ij}\ \tilde{\rho}_{ij} = \begin{bmatrix} a^2 & 0 & 0 & 0\\ 0 & b^2+c^2 & 2bc & 0\\ 0 & 2bc & b^2+c^2 & 0\\ 0 & 0 & 0 & a^2\end{bmatrix}\begin{bmatrix} a^2 & 0 & 0 & 0\\ 0 & b^2+c^2 & 2bc & 0\\ 0 & 2bc & b^2+c^2 & 0\\ 0 & 0 & 0 & a^2\end{bmatrix} \tag{5.34}$$

$$= \begin{bmatrix} a^4 & 0 & 0 & 0\\ 0 & 4b^2c^2+(b^2+c^2)^2 & 4bc(b^2+c^2) & 0\\ 0 & 4bc(b^2+c^2) & 4b^2c^2+(b^2+c^2)^2 & 0\\ 0 & 0 & 0 & a^4\end{bmatrix} \tag{5.35}$$

The eigenvalues of this matrix $\rho_{ij}\ \tilde{\rho}_{ij}$ from Eq.[5.35] is obtained as,

$$\lambda_1 = (b+c)^4,\ \lambda_2 = (b-c)^4,\ \lambda_3 = a^4,\ \lambda_4 = a^4 \tag{5.36}$$

So the concurrence, as per our definition, will be given by,

$$C_{i,j} = \max\{0,\ \sqrt{\lambda_1} - \sqrt{\lambda_2} - \sqrt{\lambda_3} - \sqrt{\lambda_4}\} \tag{5.37}$$

$$= \max\{0, (b+c)^2\ - (b-c)^2\ - a^2\ - a^2\} \tag{5.38}$$

Substituting $(b+c)^2$,$(b-c)^2$ and a^2 in terms of the correlation functions, we get,

$$\begin{aligned} C_{ij} &= \max\{0, (b+c)^2\ - (b-c)^2\ - a^2\ - a^2\} \\ &= \max\{\ 0, (\frac{1}{4} - \frac{1}{4}(G_{zz} - G_{xx} - G_{yy})) - (\frac{1}{4} - \frac{1}{4}(G_{zz} + G_{xx} + G_{yy})) - \frac{1}{2}(1 + G_{zz})\} \\ &= \max\{\ 0, \frac{1}{4} - \frac{1}{4}(G_{zz} - G_{xx} - G_{yy}) - \frac{1}{4} + \frac{1}{4}(G_{zz} + G_{xx} + G_{yy})) - \frac{1}{2}(1 + G_{zz})\} \\ &= \max\{\ 0, \frac{1}{2}(|G_{xx} + G_{yy}| - G_{zz} - 1)\} \end{aligned} \tag{5.39}$$

5.3.1.3 Numerical Verification

Now we can check numerically whether this relation gives us the same concurrence that we were getting earlier. For that, we can construct the following Fortran code.

```
program Heisenberg_model_with_fibonacci
    !For n qubit system
    !when J is varied like inverse of Fibonacci series and B is also
    varied from 0 to 1 in steps of 0.01.
    !Open chain System
    implicit none
    integer :: i, n, num, o, a, x, y, k,  differing_bits
    integer, parameter :: q = 1, t = 1, c = 1
    real*8 :: B, dB
    integer :: m
    integer, parameter :: d = 2
    real*8:: con1
    real*8 :: be1, be2
    character(len=:), allocatable :: state(:)
    integer, allocatable :: decimal(:)
```

```
15      integer, allocatable :: psi(:)
16      double precision, allocatable :: p(:), vec1(:), res1(:), vit(:),
         rho(:,:)
17      integer :: j
18      real*8, allocatable :: S1(:,:), S2(:,:),S3(:,:),S4(:,:),S5(:,:),
        H(:,:), H1(:,:)
19      integer, allocatable :: site(:)
20
21      ! Parameters for LAPACK DSYEV
22      integer :: lda, info, lwork
23      real*8, allocatable :: w(:), work(:), dw(:)
24      character*1 :: JOBZ, UPLO
25
26      ! Input values
27      print*, "Enter the number of bits you want:"
28      read*, n
29      num = 2**n ! Calculate number of states
30      m = n
31      ! Allocate arrays after determining the value of num
32      allocate(character(len=n) :: state(num))
33      allocate(integer :: decimal(num))
34      allocate(psi(n))
35      allocate(S1(num, num))
36      allocate(S2(num, num))
37      allocate(S3(num, num))
38      allocate(S4(num, num))
39      allocate(S5(num, num))
40      allocate(H(num, num))
41      allocate(H1(num, num))
42      allocate(vec1(num))
43      allocate(vit(0:num-1))
44      allocate(p(m))
45      allocate(rho(0:num-1,0:num-1))
46      allocate(site(0:d-1))
47
48      ! LAPACK parameters
49      lda = num
50      lwork = 3 * num - 1
```

```
51      allocate(real(8) :: w(lda), work(lwork))
52
53      ! Printing the possible states
54      !print*, "The states are:"
55      do i = 0, num - 1
56          call integer_binary(i, state(i + 1), n)  ! Adjusting index
    for 1-based
57          !print *, state(i + 1)
58      end do
59
60      !write(722,*) "Decimal equivalents of the states"
61
62      ! Convert binary states to decimal
63      do i = 1, num
64          decimal(i) = btod(state(i), n)
65          !write(722,*) decimal(i)
66      end do
67
68      call FibonacciSeries(m, p)
69
70      !Perform calculation based on sigma_i^x*sigma_i+1^x as S1
71      ! Initialize the S1 matrix
72          S1 = 0.0
73          do i = 1, num
74              do j = 1, num
75                  ! Initialize differing_bits counter
76                  differing_bits = 0
77                  do k = 0, n-1
78                      ! Compare the k-th bit of the i-th and j-th
    states
79                      if (btest(decimal(i), k) .neqv. btest(decimal(j)
    , k)) then
80                          differing_bits = differing_bits + 1
81                      end if
82                  end do
83                  ! Check if exactly two bits differ, and those bits
    are the specified positioned two qubits
84                  if (differing_bits == 2) then
```

```
85                    do o = 1,(n-1),1
86                        !print*, i, j, decimal(i), decimal(j)
87                        if ((btest(decimal(i), (n-o)) .neqv. btest(
   decimal(j), (n-o))) .and. &
88                            (btest(decimal(i), (n-o-1)) .neqv. btest
   (decimal(j), (n-o-1)))) then
89                            S1(i, j) = p(o)
90                            !print*, i, j, o, btest(decimal(i), (n-o
   )), btest(decimal(j), (n-o)), p(o), S1(i,j)
91                        end if
92                    end do
93                else
94                    S1(i, j) = 0.0
95                end if
96
97            end do
98        end do
99     ! do i = 1, num
100    !      do j = 1, num
101    !          write(9,*) i, j, S1(i,j)
102    !      end do
103    ! end do
104    !Perform calculation based on sigma_i^y*sigma_i+1^y as S2
105    ! Initialize the S2 matrix
106        S2 = 0.0
107        do i = 1, num
108            do j = 1, num
109             call single_digit(state(i), psi)
110                ! Initialize differing_bits counter
111                differing_bits = 0
112                do k = 0, n-1
113                    ! Compare the k-th bit of the i-th and j-th
   states
114                    if (btest(decimal(i), k) .neqv. btest(decimal(j)
   , k)) then
115                        differing_bits = differing_bits + 1
116                    end if
117                end do
```

```
118                 ! Check if exactly two bits differ, and those bits
    are the specified positions two qubits
119                 if (differing_bits == 2) then
120                     do o = 1,(n-1),1
121                         !print*, i, j, decimal(i), decimal(j)
122                         if ((btest(decimal(i), (n-o)) .neqv. btest(
    decimal(j), (n-o))) .and. &
123                             (btest(decimal(i), (n-o-1)) .neqv. btest
    (decimal(j), (n-o-1)))) then
124                              S2(i, j) = (-1)**(psi(o) + psi(o+1) +
    1)*p(o)
125                             !print*, i, j, o, btest(decimal(i), (n-o
    )), btest(decimal(j), (n-o)), p(o), S1(i,j)
126                         end if
127                     end do
128
129                 else
130                     S2(i, j) = 0.0
131                 end if
132
133             end do
134         end do
135     ! do i = 1, num
136     !     do j = 1, num
137     !         write(8,*) i, j, S2(i,j)
138     !     end do
139     ! end do
140
141
142
143
144     !Perform calculation based on sigma_i^z*sigma_i+1^z as S3
145     ! Perform calculations based on the decimal values for S3
146      do i = 1, num
147         do j = 1, num
148             call single_digit(state(i), psi)
149             if (decimal(i) /= decimal(j)) then
150                 S3(i, j) = 0.0
```

```
151             else
152                 S3(i, j) = 0.0
153                 do k = 2, n
154                     S3(i, j) = S3(i, j) + p(k-1)*(1.0 - 2.0 * psi(k
    -1)) * (1.0 - 2.0 * psi(k))
155                 end do
156             end if
157         end do
158     end do
159
160     ! Construct the S4 matrix (for sigma_x )
161     S4 = 0.0
162     do i = 1, num
163         do j = 1, num
164             ! Initialize differing_bits counter
165             differing_bits = 0
166             do k = 1, n
167                 ! Compare the k-th bit of the i-th and j-th states
168                 if (btest(decimal(i), k-1) .neqv. btest(decimal(j),
    k-1)) then
169                     differing_bits = differing_bits + 1
170                 end if
171             end do
172             ! If exactly one bit differs, set S4(i, j) = 1.0
173             if (differing_bits == 1) then
174                 S4(i, j) = 1.0
175             else
176                 S4(i, j) = 0.0
177             end if
178         end do
179     end do
180     ! Perform calculations based on the decimal values for S5
181     do i = 1, num
182         do j = 1, num
183             call single_digit(state(i), psi)
184             if (decimal(i) /= decimal(j)) then
185                 S5(i, j) = 0.0
186             else
```

```
                S5(i, j) = 0.0
                do k = 1, n
                    S5(i, j) = S5(i,j) + (1.0 - 2.0 * psi(k))
                end do
            end if
        end do
    end do

    ! do i = 1, num
    !     do j = 1, num
    !         write(7,*) i,j, S3(i,j)
    !         ! write(4,*)  S2
    !     end do
    ! end do


    !call FibonacciSeries(m, p)
    ! print *, "Inverse Fibonacci series:"
    ! do i = 1, m
    !     print *, i, "p(", i, ") = ", p(i)
    ! end do


    ! Total Hamiltonian
    do y = 0, 100, 1
            B = y * 0.01

            H =  (q*S1 + t*S2 + c*S3 ) + B * S5

             ! Eigenvalue and eigenvector determination using DSYEV
            JOBZ = 'V'
            UPLO = 'U'
            call DSYEV(JOBZ, UPLO, num, H, lda, w, work, lwork, info
   )
              vec1 = H(:, 1)
            do i = 1, num
                vit(i-1) = vec1(i)
            end do
```

```
224
225
226         !Concurrence
227         do i = 0, n-1
228             do j = i+1, n-1
229                 if( i /= (n-1)) then
230                 call Correlation_Con(n,i,j,vit,con1)
231                 write(74,*) B , con1
232                 else
233                     EXIT
234                 end if
235             end do
236         end do
237     end do
238
239
240
241
242     deallocate(state)
243     deallocate(decimal)
244     deallocate(psi)
245     deallocate(S1)
246     deallocate(S2)
247     deallocate(S3)
248     deallocate(S4)
249     deallocate(S5)
250     deallocate(H)
251     deallocate(H1)
252     deallocate(p)
253     deallocate(rho)
254
255 contains
256     function btod(state, n) result(decimal)
257         implicit none
258         integer, intent(in) :: n
259         character(len=n) :: state
260         integer :: decimal, i
261         decimal = 0
```

```
262         do i = 1, n
263             if (state(i:i) == '1') then
264                 decimal = decimal + 2**(n-i)
265             end if
266         end do
267     end function btod
268 
269     subroutine integer_binary(num, binary_string, n)
270         implicit none
271         integer :: num
272         integer, intent(in) :: n
273         integer :: i, temp_num
274         character(len=n), intent(out) :: binary_string
275         temp_num = num
276         binary_string = repeat('0', n)
277         do i = 0, n - 1
278             if (mod(temp_num, 2) == 1) then
279                 binary_string(n-i:n-i) = '1'
280             else
281                 binary_string(n-i:n-i) = '0'
282             end if
283             temp_num = temp_num / 2
284         end do
285     end subroutine integer_binary
286 
287     subroutine single_digit(A, psi)
288         implicit none
289         character(len=*), intent(in) :: A
290         integer, intent(out) :: psi(:)
291         integer :: i
292         do i = 1, len(A)
293             if (A(i:i) == '0') then
294                 psi(i) = 0
295             else if (A(i:i) == '1') then
296                 psi(i) = 1
297             else
298                 print *, "Invalid input in single_digit subroutine"
299             end if
```

```
300         end do
301     end subroutine single_digit
302
303     subroutine FibonacciSeries(m, reciprocal)
304         implicit none
305         integer :: i
306         integer :: m
307         double precision :: fib(m)
308         double precision :: reciprocal(m)
309         if (m <= 0) then
310             print*, "Number of terms must be greater than 0."
311             stop
312         end if
313         ! Initialize the first two terms
314         fib(1) = 1
315         if (n > 1) then
316             fib(2) = 1
317         end if
318         ! Generate the Fibonacci series
319         do i = 3, m
320             fib(i) = fib(i-1) + fib(i-2)
321         end do
322         ! Compute reciprocals
323         do i = 1, m
324             reciprocal(i) = 1.0d0 / real(fib(i))
325         end do
326         !print*, fib
327     end subroutine FibonacciSeries
328
329     subroutine PTRVR(s, s1, site, vin, rdm)
330         implicit none
331         integer :: s, s1, s2
332         real*8 :: trace
333         real*8, dimension(0:2**s-1) :: vin
334         real*8, dimension(0:2**s1-1, 0:2**s1-1) :: rdm
335         integer :: ii, i1, a, i0, ia, j1, oo, k1, x1, y1, j, t1, t2,
    z1
336         integer, dimension(0:s1-1) :: site, site2
```

```
337         integer, dimension(0:s1-1) :: bin
338         integer, dimension(0:(2**(s-s1))*(2**s1)-1) :: ind
339         s2 = s - s1
340         x1 = 0
341         y1 = 0
342         ind = 0
343             do j1 = 2**s1-1, 0, -1
344                 do ii = 0, 2**s-1
345                     a = ii
346                     do i1 = 0, s1-1
347                         i0 = site(i1)
348                         call DTOBONEBIT(a, ia, i0, s)
349                         site2(i1) = ia
350                     end do
351                     call DTOB(j1, bin, s1)
352                     oo = 1
353                     do k1 = 0, s1-1
354                         oo = oo * (bin(k1) - site2(k1))
355                     end do
356                     if (abs(oo) == 1) then
357                         ind(x1) = ii
358                         x1 = x1 + 1
359                     end if
360                 end do
361             end do
362
363             rdm = 0.0d0
364             do t1 = 0, 2**s1-1
365                 do t2 = 0, 2**s1-1
366                     do z1 = 0, 2**s2-1
367                         rdm(t1, t2) = rdm(t1, t2) + vin(ind((2**s2)
    * t1 + z1)) * vin(ind((2**s2) * t2 + z1))
368                     end do
369                 end do
370             end do
371         end subroutine
372         subroutine PTRDMR(s,s1,site,rho,rdm)
373         implicit none
```

```
374         integer::s, s1,s2
375         real*8::trace
376         real*8,dimension(0:2**s-1,0:2**s-1)::rho
377         real*8,dimension(0:2**s1-1,0:2**s1-1)::rdm
378         integer::ii,i1,a,i0,ia,j1,oo,k1,x1,y1,j,t1,t2,z1,i
379         integer,dimension(0:s1-1)::site,site2
380         integer,dimension(0:s1-1)::bin
381         integer,dimension(0:(2**(s-s1))*(2**s1)-1)::ind
382         s2=s-s1
383         x1=0
384         y1=0
385         ind=0
386         do j1=2**s1-1,0,-1
387             do ii=0,2**s-1
388                 a=ii
389                 do i1=0,s1-1,1
390                     i0=site(i1)
391                     call DTOBONEBIT(a,ia,i0,s)
392                     site2(i1)=ia
393                 end do
394                 call DTOB(j1,bin,s1)
395                 oo=1
396                 do k1=0,s1-1,1
397                     oo=oo*(bin(k1)-site2(k1))
398                 end do
399                 if(abs(oo)==1) then
400                     ind(x1)=ii
401                     x1=x1+1
402                 end if
403             end do
404         end do
405         rdm=0.0d0
406         do t1=0,2**s1-1,1
407             do t2=0,2**s1-1,1
408                 do z1=0,2**s2-1,1
409                     rdm(t1,t2)=rdm(t1,t2)+ &
410                     rho(ind((2**s2)*t1+z1),ind((2**s2)*t2+z1))
411                 end do
```

```
412             end do
413         end do
414     end subroutine
415         subroutine DTOB(m, tt, s)
416             implicit none
417             integer :: s
418             integer, dimension(0:s-1) :: tt
419             integer :: m, k, a2
420             tt = 0
421             a2 = m
422             do k = 0, s-1
423                 tt(s-k-1) = mod(a2, 2)
424                 a2 = a2 / 2
425                 if (a2 == 0) exit
426             end do
427         end subroutine
428          subroutine DTOBONEBIT(m, ia, i0, s)
429             implicit none
430             integer*4 :: s
431             integer :: tt(0:s-1)
432             integer :: m, ia, i0
433             call DTOB(m, tt, s)
434             ia = tt(i0)
435         end subroutine
436
437        subroutine Correlation_Con(s,i0,j0,vin,con)
438         implicit none
439         integer::s,i0,j0, s2 ,site(0:1),i, site1(0:0)
440         real*8::vin(0:2**s-1)
441         real*8,dimension(0:3,0:3)::rdm,MM1,MM2,MM3,MM4
442         real*8,dimension(0:1,0:1):: rdm1, M
443         real*8::c2,const,sigmay(0:3,0:3),sigmax(0:3,0:3),sigmaz
    (0:3,0:3)
444         real*8::sigx(0:1,0:1)
445         real*8:: con, Gxx, Gyy, Gzz, M_bar
446
447         s2= s-2
448         !Define sigmax, sigmay, sigmaz
```

```
449         sigmax(0,:)=(/0.0,0.0,0.0,1.0/)
450         sigmax(1,:)=(/0.0,0.0,1.0,0.0/)
451         sigmax(2,:)=(/0.0,1.0,0.0,0.0/)
452         sigmax(3,:)=(/1.0,0.0,0.0,0.0/)
453
454         sigmay(0,:)=(/0.0,0.0,0.0,-1.0/)
455         sigmay(1,:)=(/0.0,0.0,1.0,0.0/)
456         sigmay(2,:)=(/0.0,1.0,0.0,0.0/)
457         sigmay(3,:)=(/-1.0,0.0,0.0,0.0/)
458
459         sigmaz(0,:)=(/1.0,0.0,0.0,0.0/)
460         sigmaz(1,:)=(/0.0,-1.0,0.0,0.0/)
461         sigmaz(2,:)=(/0.0,0.0,-1.0,0.0/)
462         sigmaz(3,:)=(/0.0,0.0,0.0,1.0/)
463
464         sigx(0,:)=(/0.0,1.0/)
465         sigx(1,:)=(/1.0,0.0/)
466
467
468         site(0)=i0
469         site(1)=j0
470         call PTRVR(s,2,site,vin,rdm)
471         write(3,*) rdm
472
473         site1(0)=i0
474         call PTRDMR(s2,1,site1,rdm,rdm1)
475
476         MM1=matmul(rdm,sigmax)
477         MM2=matmul(rdm,sigmay)
478         MM3=matmul(rdm,sigmaz)
479         M = matmul(sigx,rdm1)
480
481         !Correlation functions
482         Gxx = 0.0
483         do i = 0, 3
484             Gxx = Gxx + MM1(i,i)
485         end do
486
```

```
487         Gyy = 0.0
488         do i = 0, 3
489             Gyy = Gyy + MM2(i,i)
490         end do
491 
492         Gzz = 0.0
493         do i = 0, 3
494             Gzz = Gzz + MM3(i,i)
495         end do
496         M_bar = 0.0
497         do i = 0, 1
498             M_bar = M_bar + M(i,i)
499         end do
500         c2=0
501         con=0
502         c2 = 0.5d0*(abs(Gxx+Gyy)-sqrt(((1+Gzz)*(1+Gzz))-4*M_bar))
503         print*, c2
504         if(c2>0)then
505             con=c2
506         else
507            con=0
508         end if
509     end subroutine
510 end program Heisenberg_model_with_fibonacci
```

By implementing the above code, we can check the consistency of the Eq.[5.39], which relates the concurrence with the correlation function. We can also plot the concurrence obtained from this equation and compare it with the concurrence obtained earlier, which is shown below.

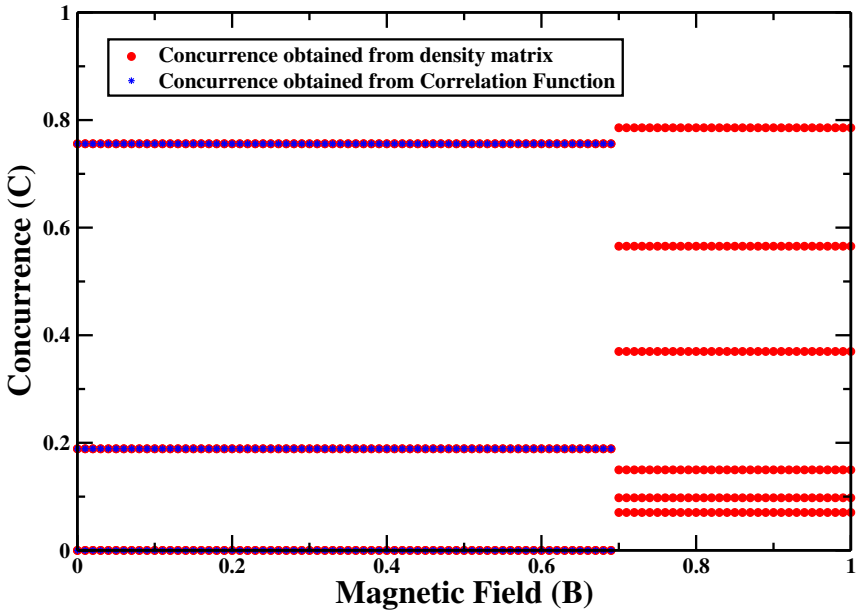


Figure 5.6: Concurrences obtained from correlation function and that obtained earlier for 4 qubit open chain Heisenberg-like spin chain system.

According to the **Feynman-Hellmann theorem** [24], we can say that since there is a discontinuity in the ground state eigenvalue due to the quantum phase transition at the 0.70 unit magnetic field it results in the change of the ground state eigenvector of the system resulting in the discontinuity in concurrence which we have discussed in details above.

Note: Feynman-Hellmann Theorem

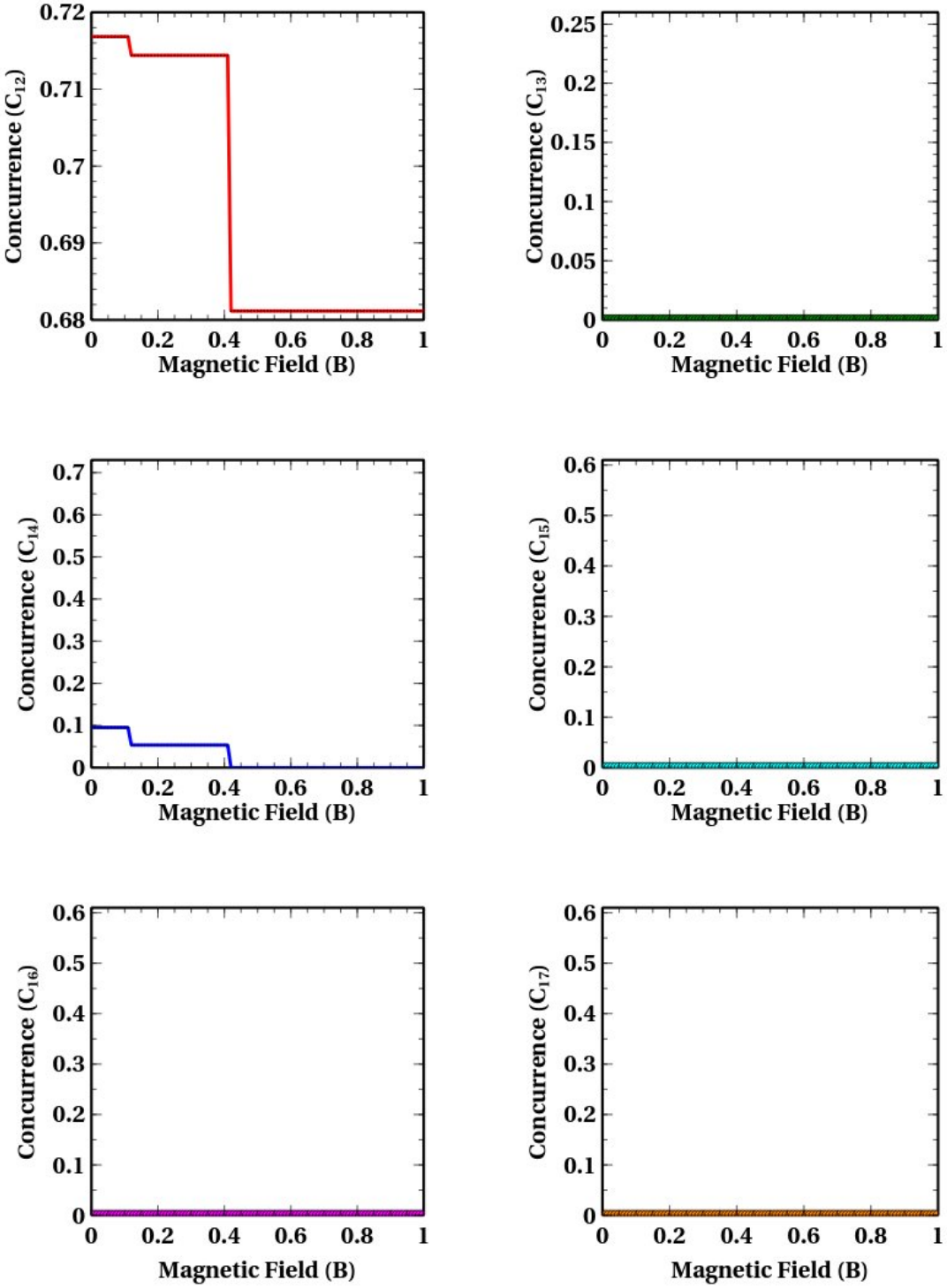
In quantum mechanics, the Feynman-Hellmann theorem relates the derivative of the total energy with respect to a parameter to the expectation value of the derivative of the Hamiltonian with respect to that same parameter. Suppose H_λ be the Hamiltonian dependent on some parameter λ and E_λ be the eigenvalue corresponding to the state ψ_λ of the Hamiltonian then,

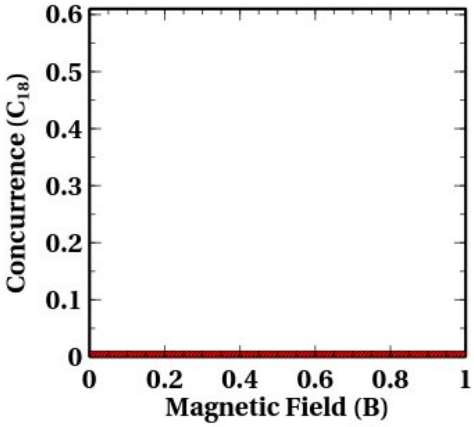
$$\frac{dE_\lambda}{d\lambda} = \langle\psi_\lambda|\frac{\partial H_\lambda}{\partial\lambda}|\psi_\lambda\rangle$$

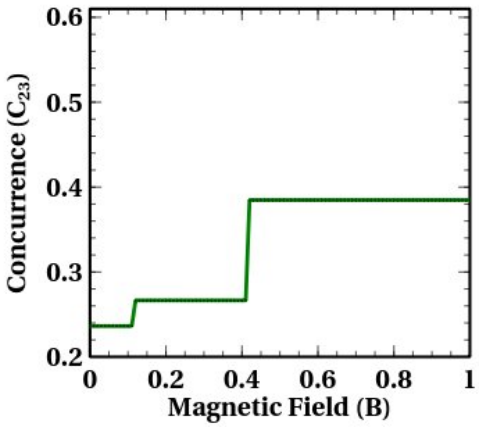
We had also studied the concurrence variation for an 8-qubit open chain system. Here we can see the significant step-wise discontinuity in the nearest neighbour

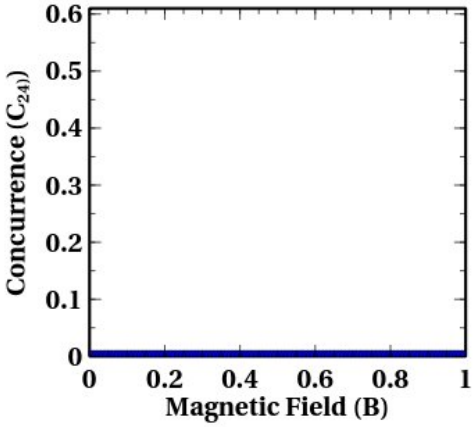
concurrence, whereas the concurrence corresponding to the other qubit pair is zero.

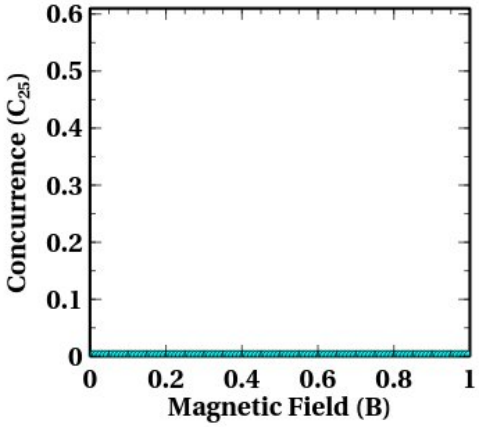
5.3.1.4 Concurrences for 8 qubit open Heisenberg-Fibonacci spin chain system

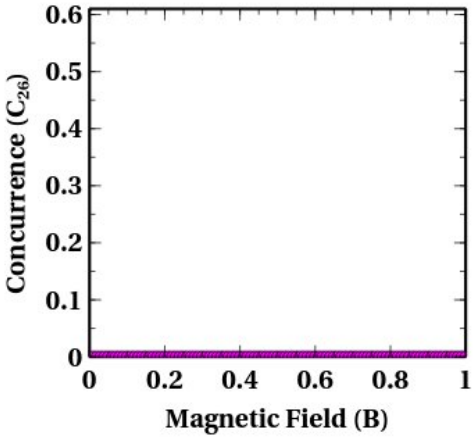


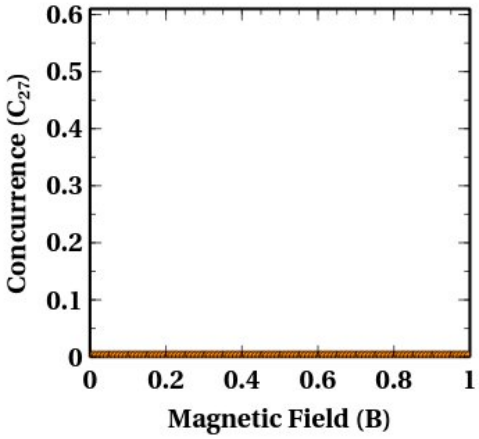

Concurrence (C_{18})
Magnetic Field (B)

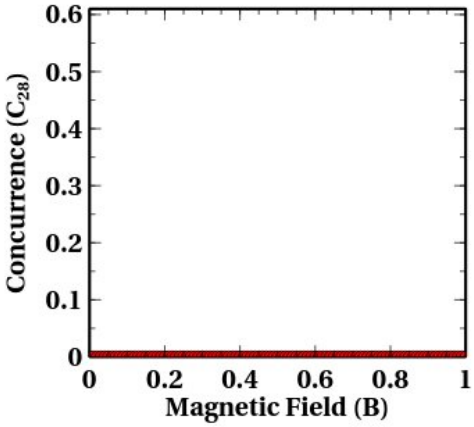

Concurrence (C_{23})
Magnetic Field (B)

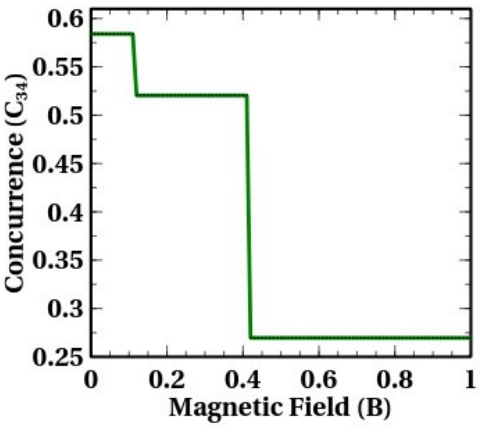

Concurrence (C_{24})
Magnetic Field (B)

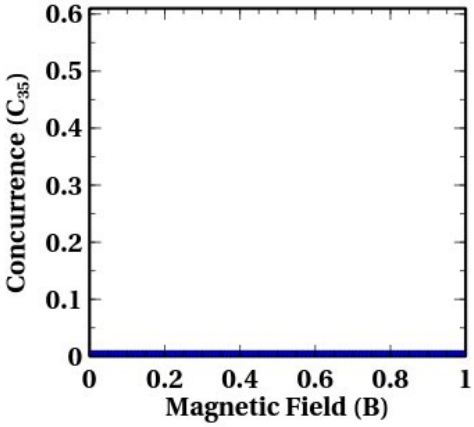

Concurrence (C_{25})
Magnetic Field (B)

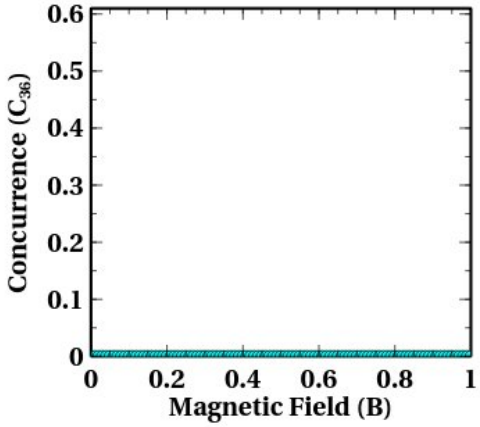

Concurrence (C_{26})
Magnetic Field (B)

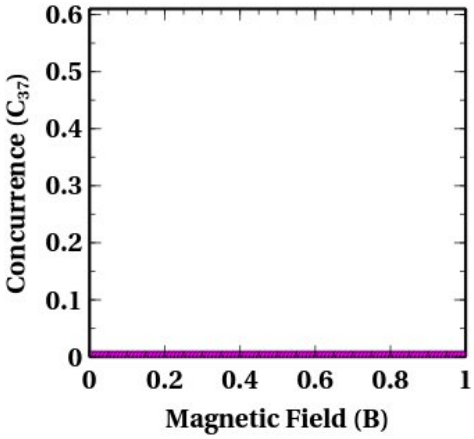

Concurrence (C_{27})
Magnetic Field (B)

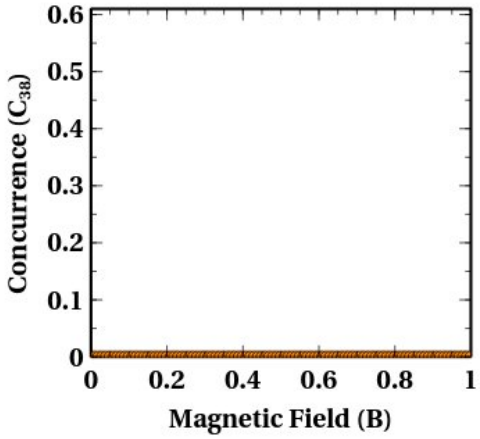

Concurrence (C_{28})
Magnetic Field (B)

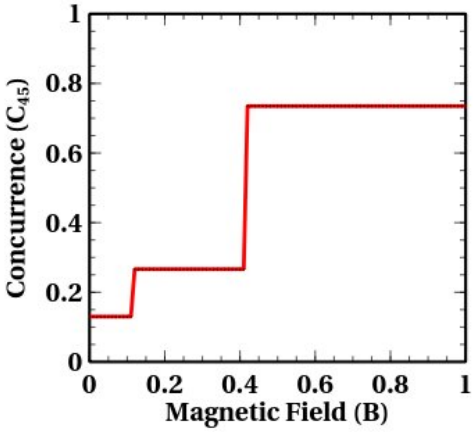

Concurrence (C_{34})
Magnetic Field (B)

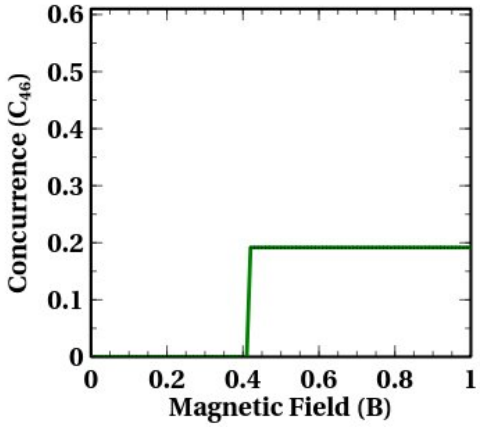

Concurrence (C_{35})
Magnetic Field (B)

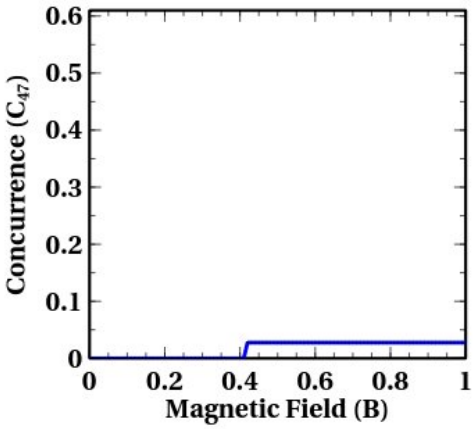

Concurrence (C_{36})
Magnetic Field (B)

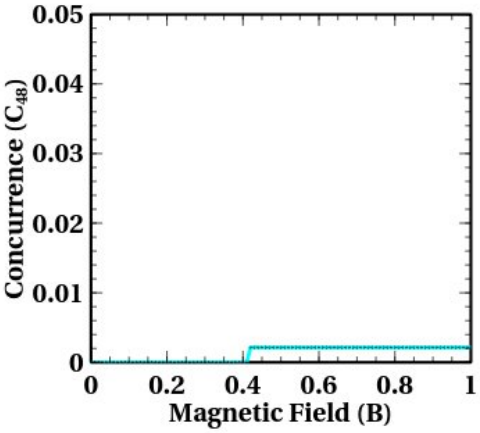

Concurrence (C_{37})
Magnetic Field (B)

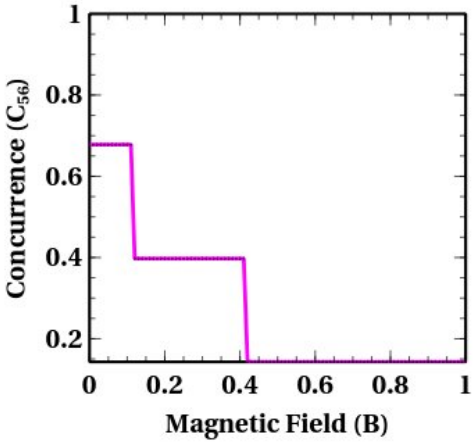

Concurrence (C_{38})
Magnetic Field (B)

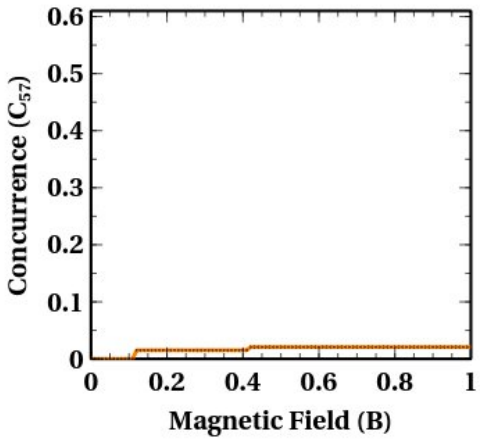

Concurrence (C_{45})
Magnetic Field (B)

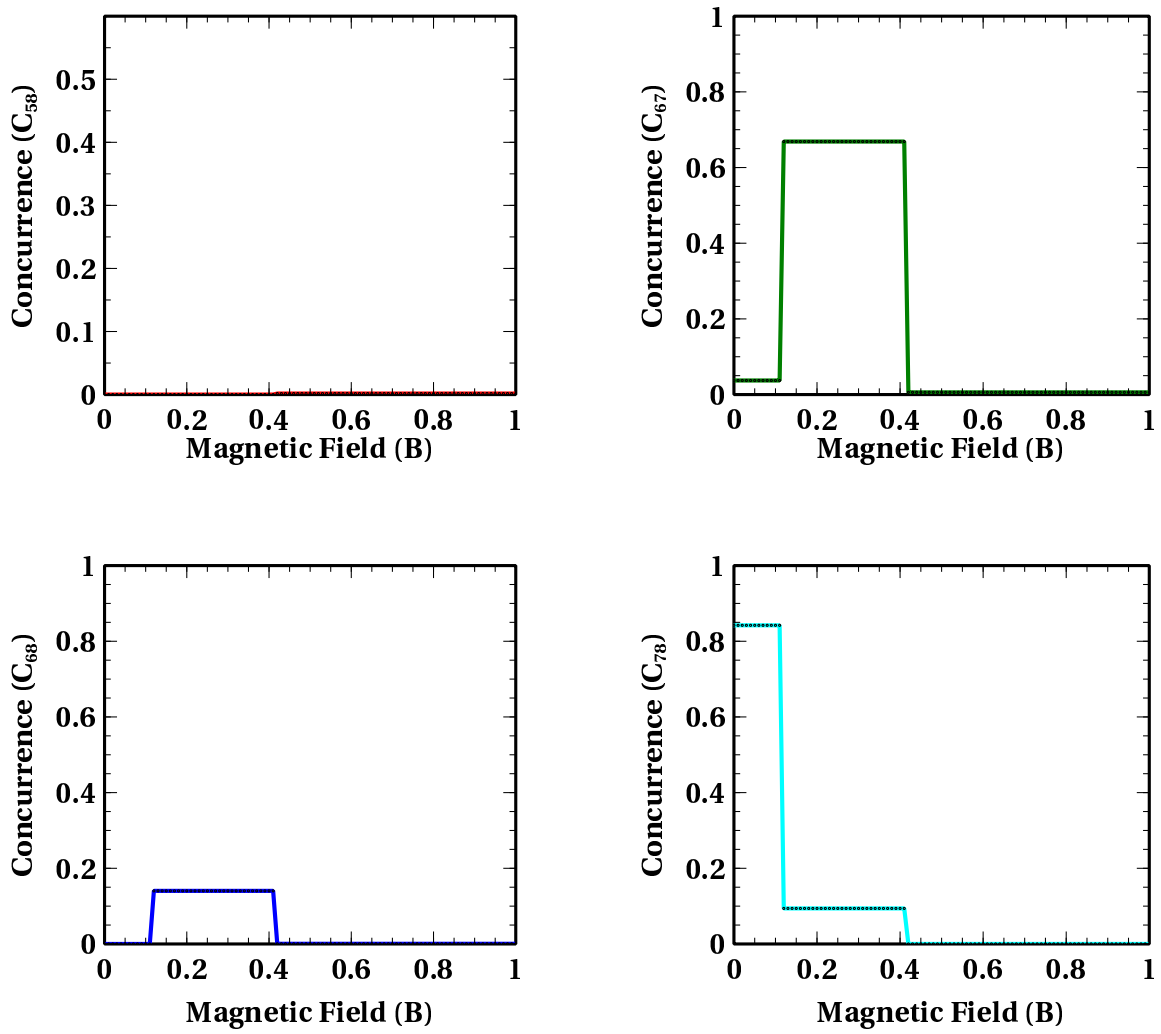

Concurrence (C_{46})
Magnetic Field (B)


Concurrence (C_{47})
Magnetic Field (B)


Concurrence (C_{48})
Magnetic Field (B)


Concurrence (C_{56})
Magnetic Field (B)


Concurrence (C_{57})
Magnetic Field (B)



5.3.1.5 Case(2): Periodic chain System

To study the concurrences between different qubit pairs for a periodic chain system, we can implement the following Fortran code.

```
program Heisenberg_model_with_fibonacci_open_Concurrence
!For n qubit periodic chain system
!Program of generalised Ising Model when J is varied like inverse of
    Fibonacci series and B is also varied
implicit none
integer :: i, n, num, o, a, x, y, k,  differing_bits
integer, parameter :: q = 1, t = 1, c = 1
integer, parameter :: d = 3
```

```
real*8 :: B
real :: con, be2, con1
real*8 ::  be1
integer :: l
character(len=:), allocatable :: state(:)
integer, allocatable :: decimal(:)
integer, allocatable :: psi(:)
double precision, allocatable :: p(:), vec1(:), res1(:), vit(:)
integer :: j
real*8, allocatable :: S1(:,:), S2(:,:),S3(:,:),S4(:,:), S5(:,:), H
    (:,:), H1(:,:), rho(:,:)
integer, allocatable :: site(:)

! Parameters for LAPACK DSYEVX
integer :: LDA, info, LWORK, LDZ, M
real*8, allocatable :: Z(:,:)
integer, parameter :: IL = 1 , IU = 2
real*8, allocatable :: w(:), work(:)
character*1 :: JOBZ, UPLO
double precision :: VL, VU
integer, allocatable :: IFAIL(:), IWORK(:)
double precision, parameter :: ABSTOL = 0.0d0
character*1 :: RANGE   !'A' for computing all eigenvalues and 'I'
    for IL-th through IU-th eigenvalue

! Input values
print*, "Enter the number of bits you want:"
read*, n
num = 2**n ! Calculate number of states
l = n
! Allocate arrays after determining the value of num
allocate(character(len=n) :: state(num))
allocate(integer :: decimal(num))
allocate(psi(n))
allocate(S1(num, num))
allocate(S2(num, num))
allocate(S3(num, num))
allocate(S4(num, num))
```

```
allocate(S5(num, num))
allocate(H(num, num))
allocate(H1(num, num))
allocate(p(m))
allocate(vec1(num))
allocate(vit(0:num-1))
allocate(res1(num))
allocate(rho(num,num))
allocate(site(0:d-1))

! LAPACK parameters
LDA = num
LWORK = 8 * num + 1
M = IU-IL+1
LDZ = num
allocate(real(8) ::  w(LDA), work(LWORK))
allocate(integer :: IWORK(5*num), IFAIL(num))
allocate(real(8) :: Z(num,num))

! Printing the possible states
print*, "The states are:"
do i = 0, num - 1
    call integer_binary(i, state(i + 1), n)  ! Adjusting index for
   1-based
        print *, state(i + 1)
end do


! Convert binary states to decimal
do i = 1, num
    decimal(i) = btod(state(i), n)
   !write(722,*) decimal(i)
end do

call FibonacciSeries(l, p)

!Perform calculation based on sigma_i^x*sigma_i+1^x as S1
! Initialize the S1 matrix
```

```
S1 = 0.0
do i = 1, num
    do j = 1, num
    ! Initialize differing_bits counter
    differing_bits = 0
    do k = 0, n-1
    ! Compare the k-th bit of the i-th and j-th states
    if (btest(decimal(i), k) .neqv. btest(decimal(j), k)) then
        differing_bits = differing_bits + 1
    end if
    end do
    ! Check if exactly two bits differ, and those bits are the specified positioned two qubits
    if (differing_bits == 2) then
        do o = 1,(n-1),1
            !print*, i, j, decimal(i), decimal(j)
            if ((btest(decimal(i), (n-o)) .neqv. btest(decimal(j), (n-o))) .and. &
            (btest(decimal(i), (n-o-1)) .neqv. btest(decimal(j), (n-o-1)))) then
            S1(i, j) = p(o)
            !print*, i, j, o, btest(decimal(i), (n-o)), btest(decimal(j), (n-o)), p(o), S1(i,j)
            end if
        end do
    else
        S1(i, j) = 0.0
    end if

    end do
end do

!Perform calculation based on sigma_i^y*sigma_i+1^y as S2
! Initialize the S2 matrix
S2 = 0.0
do i = 1, num
    do j = 1, num
        call single_digit(state(i), psi)
```

```
115         ! Initialize differing_bits counter
116         differing_bits = 0
117         do k = 0, n-1
118         ! Compare the k-th bit of the i-th and j-th states
119         if (btest(decimal(i), k) .neqv. btest(decimal(j), k)) then
120         differing_bits = differing_bits + 1
121         end if
122         end do
123     ! Check if exactly two bits differ, and those bits are the
    specified positioned two qubits
124     if (differing_bits == 2) then
125         do o = 1,(n-1),1
126         if ((btest(decimal(i), (n-o)) .neqv. btest(decimal(j), (n-o)
    )) .and. &
127         (btest(decimal(i), (n-o-1)) .neqv. btest(decimal(j), (n-o-1)
    ))) then
128             S2(i, j) = (-1)**(psi(o) + psi(o+1) + 1)*p(o)
129             !print*, i, j, o, btest(decimal(i), (n-o)), btest(
    decimal(j), (n-o)), p(o), S1(i,j)
130         end if
131         end do
132         if ((btest(decimal(i), (n-1)) .neqv. btest(decimal(j), (n-1)
    )) .and. &
133         (btest(decimal(i), 0) .neqv. btest(decimal(j), 0))) then
134         S2(i, j) = (-1)**(psi(n-2) + psi(n) + 1)*p(n)
135         end if
136         else
137             S2(i, j) = 0.0
138         end if
139     end do
140 end do
141 !Perform calculation based on sigma_i^z*sigma_i+1^z as S3
142 ! Perform calculations based on the decimal values for S3
143 do i = 1, num
144     do j = 1, num
145         call single_digit(state(i), psi)
146         if (decimal(i) /= decimal(j)) then
147             S3(i, j) = 0.0
```

```
148         else
149             S3(i, j) = 0.0
150             do k = 1, n - 1
151                 S3(i, j) = S3(i, j) + p(k)*(1.0 - 2.0 * psi(k)) *
    (1.0 - 2.0 * psi(k + 1))
152             end do
153             S3(i, j) = S3(i, j) + p(n)*(1.0 - 2.0 * psi(n)) * (1.0 -
     2.0 * psi(1))
154         end if
155     end do
156 end do
157
158 ! Construct the S4 matrix (for sigma_x )
159 S4 = 0.0
160 do i = 1, num
161     do j = 1, num
162         ! Initialize differing_bits counter
163         differing_bits = 0
164         do k = 1, n
165         ! Compare the k-th bit of the i-th and j-th states
166             if (btest(decimal(i), k-1) .neqv. btest(decimal(j), k-1)
    ) then
167                 differing_bits = differing_bits + 1
168             end if
169         end do
170         ! If exactly one bit differs, set S4(i, j) = 1.0
171         if (differing_bits == 1) then
172             S4(i, j) = 1.0
173         else
174             S4(i, j) = 0.0
175         end if
176     end do
177 end do
178
179 !constructing the matrix S5 When the magnetic field is along the z
    direction (for sigma_z )
180 do i = 1, num
181     do j = 1, num
```

```
182     call single_digit(state(i), psi)
183     if (decimal(i) /= decimal(j)) then
184         S5(i, j) = 0.0
185     else
186         S5(i, j) = 0.0
187     do k = 1, n
188         S5(i, j) = S5(i,j) + (1.0 - 2.0 * psi(k))
189     end do
190     end if
191     end do
192 end do
193
194 ! Total Hamiltonian
195 do y = 0, 100, 1
196     B = y * 0.01
197     H =  (q*S1 + t*S2 + c*S3 ) + B * S4
198     ! Ground State and 1st excited state Eigenvalue and eigenvector
    determination using DSYEVX
199     JOBZ = 'V'
200     UPLO = 'U'
201     RANGE = 'I'
202     call DSYEVX (JOBZ, RANGE, UPLO, num, H, LDA, VL, VU, IL, IU,
    ABSTOL, M, W, Z, LDZ, WORK, LWORK, IWORK, IFAIL, INFO)
203     vec1 = Z(:, 1)
204     do i = 1, num
205         vit(i-1) = vec1(i)
206     end do
207     write(2,*) vit
208     if (info == 0) then
209         do a = 1, LDA
210             write(17,*) B, w(a) !plot for magfield vs eigenvalue
211         end do
212     else
213         write(15,*) 'Error in DSYEV, INFO=', info
214     end if
215
216     !Concurrence
217     do i = 0, n-1
```

```
218         do j = i+1, n-1
219             if( i /= (n-1)) then
220             call CONVR(n,i,j,vit,con1)
221             write(8001,*) B , con1
222             else
223                 EXIT
224             end if
225         end do
226     end do
227 end do
228 
229 deallocate(state)
230 deallocate(decimal)
231 deallocate(psi)
232 deallocate(S1)
233 deallocate(S2)
234 deallocate(S3)
235 deallocate(S4)
236 deallocate(S5)
237 deallocate(H)
238 deallocate(H1)
239 deallocate(p)
240 deallocate(rho)
241 deallocate(vit)
242 deallocate(vec1)
243 deallocate(site)
244 
245 contains
246 function btod(state, n) result(decimal)
247     implicit none
248     integer, intent(in) :: n
249     character(len=n) :: state
250     integer :: decimal, i
251     decimal = 0
252     do i = 1, n
253         if (state(i:i) == '1') then
254             decimal = decimal + 2**(n-i)
255         end if
```

```
    end do
end function btod

    subroutine integer_binary(num, binary_string, n)
        implicit none
        integer :: num
        integer, intent(in) :: n
        integer :: i, temp_num
        character(len=n), intent(out) :: binary_string
        temp_num = num
        binary_string = repeat('0', n)
        do i = 0, n - 1
            if (mod(temp_num, 2) == 1) then
                binary_string(n-i:n-i) = '1'
            else
                binary_string(n-i:n-i) = '0'
            end if
            temp_num = temp_num / 2
        end do
    end subroutine integer_binary

    subroutine single_digit(A, psi)
        implicit none
        character(len=*), intent(in) :: A
        integer, intent(out) :: psi(:)
        integer :: i
        do i = 1, len(A)
            if (A(i:i) == '0') then
                psi(i) = 0
            else if (A(i:i) == '1') then
                psi(i) = 1
            else
                print *, "Invalid input in single_digit subroutine"
            end if
        end do
    end subroutine single_digit

    subroutine FibonacciSeries(m, reciprocal)
```

```
294         implicit none
295         integer :: i
296         integer :: m
297         double precision :: fib(m)
298         double precision :: reciprocal(m)
299         if (m <= 0) then
300             print*, "Number of terms must be greater than 0."
301             stop
302         end if
303         ! Initialize the first two terms
304         fib(1) = 1
305         if (n > 1) then
306             fib(2) = 1
307         end if
308         ! Generate the Fibonacci series
309         do i = 3, m
310             fib(i) = fib(i-1) + fib(i-2)
311         end do
312         ! Compute reciprocals
313         do i = 1, m
314             reciprocal(i) = 1.0d0 / real(fib(i))
315         end do
316         !print*, fib
317     end subroutine FibonacciSeries
318     subroutine CONVR(s,i0,j0,vin,con)
319         implicit none
320         integer::s,i0,j0 ,site(0:1),i
321         real*8::vin(0:2**s-1),evs
322         real*8,dimension(0:3,0:3)::rdm,MM1,MM2,MM3,MM4
323         INTEGER::INFO, LDA, LWORK, NN,LDVL,LDVR
324         parameter(NN=4,LDA=NN,LWORK=4*NN,LDVL=NN,LDVR=NN)
325         real*8::WORK(0:LWORK-1),AA2(0:LDA-1,0:NN-1)
326         real*8::VL(0:NN-1,0:NN-1),VR(0:NN-1,0:NN-1)
327         real*8::WR(0:NN-1),WI(0:NN-1)
328         real*8::ev(NN),c2,x11,const,sigmay(0:3,0:3)
329         real :: con
330         sigmay(0,:)=(/0.0,0.0,0.0,-1.0/)
331         sigmay(1,:)=(/0.0,0.0,1.0,0.0/)
```

```
332         sigmay(2,:)=(/0.0,1.0,0.0,0.0/)
333         sigmay(3,:)=(/-1.0,0.0,0.0,0.0/)
334         site(0)=i0
335         site(1)=j0
336         call PTRVR(s,2,site,vin,rdm)
337         write(3,*) rdm
338         MM1=matmul(rdm,sigmay)
339         MM2=matmul(rdm,sigmay)
340         MM3=matmul(MM1,MM2)
341         call dgeev('N', 'N', NN, MM3, LDA, WR, &
342             WI, VL, LDVL, VR, LDVR, WORK, LWORK, INFO)
343         do i=1,NN,1
344             if (WR(i-1) .lt. 0.0000000000001) then
345                 WR(i-1)=0.0d0
346             end if
347                 ev(i)=WR(i-1)
348         end do
349         do i=2,4
350             if(ev(i)>=ev(1))then
351                 evs=ev(i)
352                 ev(i)=ev(1)
353                 ev(1)=evs
354             end if
355         end do
356         do i=3,4
357             if(ev(i)>=ev(2))then
358                 evs=ev(i)
359                 ev(i)=ev(2)
360                 ev(2)=evs
361             end if
362         end do
363         do i=4,4
364             if(ev(i)>=ev(3))then
365                 evs=ev(i)
366                 ev(i)=ev(3)
367                 ev(3)=evs
368             end if
369         end do
```

```
370         c2=0
371         con=0
372         c2=dsqrt(ev(1))-dsqrt(ev(2))-dsqrt(ev(3))-dsqrt(ev(4))
373         ! print*, c2
374         if(c2>0)then
375             con=c2
376         else
377            con=0
378         end if
379      end subroutine
380     subroutine PTRVR(s, s1, site, vin, rdm)
381         implicit none
382         integer :: s, s1, s2
383         real*8 :: trace
384         real*8, dimension(0:2**s-1) :: vin
385         real*8, dimension(0:2**s1-1, 0:2**s1-1) :: rdm
386         integer :: ii, i1, a, i0, ia, j1, oo, k1, x1, y1, j, t1, t2,
    z1
387         integer, dimension(0:s1-1) :: site, site2
388         integer, dimension(0:s1-1) :: bin
389         integer, dimension(0:(2**(s-s1))*(2**s1)-1) :: ind
390         s2 = s - s1
391         x1 = 0
392         y1 = 0
393         ind = 0
394         do j1 = 2**s1-1, 0, -1
395             do ii = 0, 2**s-1
396                 a = ii
397                 do i1 = 0, s1-1
398                     i0 = site(i1)
399                     call DTOBONEBIT(a, ia, i0, s)
400                     site2(i1) = ia
401                 end do
402                 call DTOB(j1, bin, s1)
403                 oo = 1
404                 do k1 = 0, s1-1
405                     oo = oo * (bin(k1) - site2(k1))
406                 end do
```

```
407                 if (abs(oo) == 1) then
408                     ind(x1) = ii
409                     x1 = x1 + 1
410                 end if
411             end do
412         end do
413 
414             rdm = 0.0d0
415             do t1 = 0, 2**s1-1
416                 do t2 = 0, 2**s1-1
417                     do z1 = 0, 2**s2-1
418                         rdm(t1, t2) = rdm(t1, t2) + vin(ind((2**s2)
    * t1 + z1)) * vin(ind((2**s2) * t2 + z1))
419                     end do
420                 end do
421             end do
422         end subroutine
423 
424         subroutine DTOBONEBIT(m, ia, i0, s)
425             implicit none
426             integer*4 :: s
427             integer :: tt(0:s-1)
428             integer :: m, ia, i0
429             call DTOB(m, tt, s)
430             ia = tt(i0)
431         end subroutine
432 
433         subroutine DTOB(m, tt, s)
434             implicit none
435             integer :: s
436             integer, dimension(0:s-1) :: tt
437             integer :: m, k, a2
438             tt = 0
439             a2 = m
440             do k = 0, s-1
441                 tt(s-k-1) = mod(a2, 2)
442                 a2 = a2 / 2
443                 if (a2 == 0) exit
```

```
444                end do
445             end subroutine
446
447 end program Heisenberg_model_with_fibonacci_open_concurrence
```

At first, we shall study the three-qubit periodic chain system for which we took the value of $n = 3$ in the above code. We have obtained the variations of concurrences as follows.

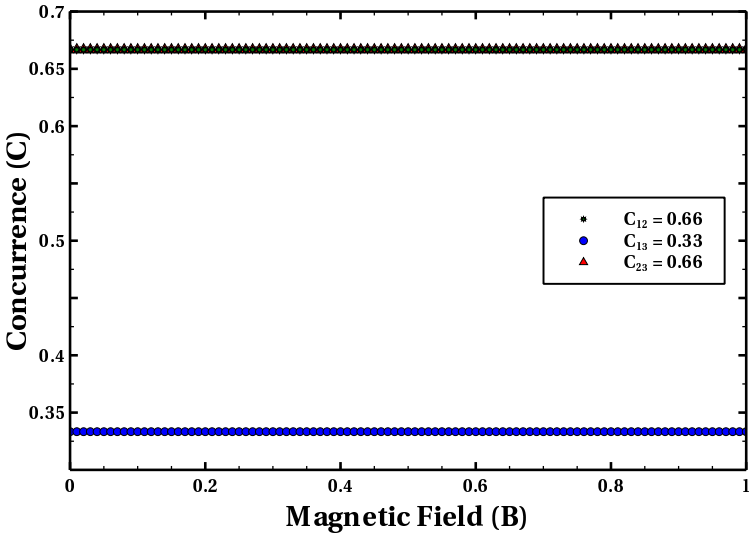
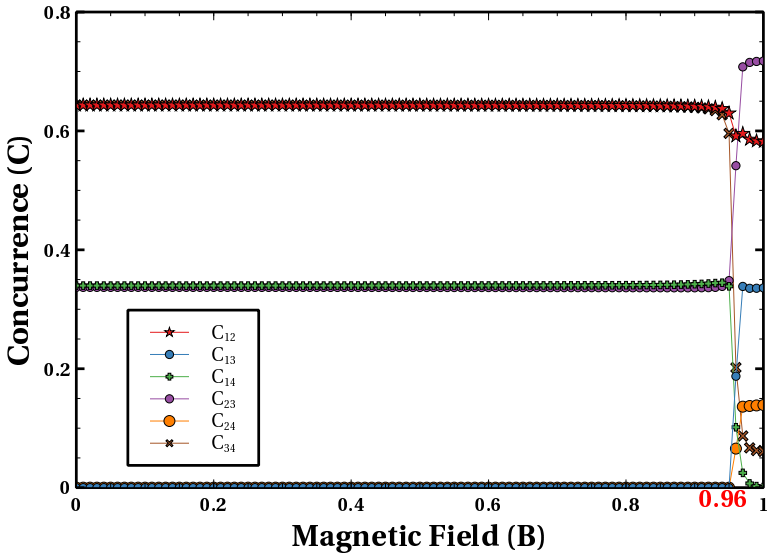
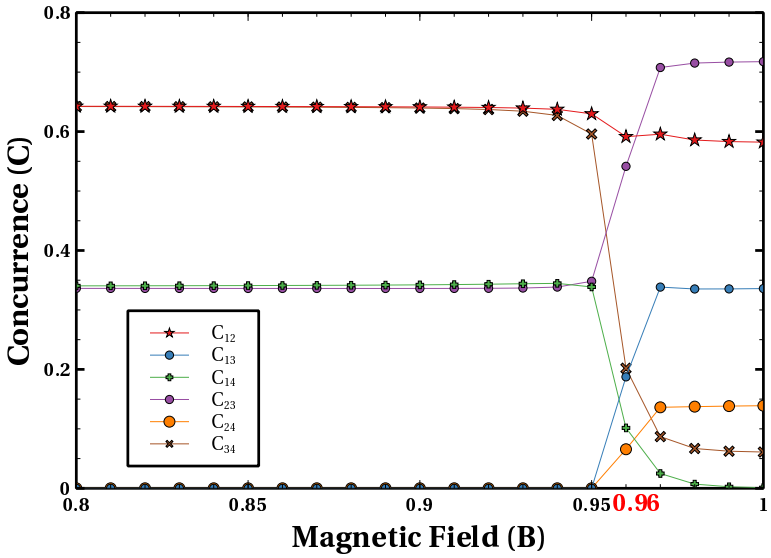


Figure 5.7: Variation of Concurrences with magnetic field for three-qubit periodic chain Heisenberg-Fibonacci system.

It is observed that for the three-qubit periodic chain system, the concurrences are continuous also constant with respect to the change in the magnetic field. its value as calculated are $C_{12} = 0.66$,C_{13}=0.33 and $C_{23} = 0.66$ respectively indicating the larger entanglement between the qubit pairs (1,2) and (2,3) and relatively small entanglement for the qubit pair (1,3). Let us go for the higher qubit system, we can see the variations of the concurrences for the qubit pairs (1,2), (1,3), (1,4), (2,3), (2,4), and (3,4) respectively for a four-qubit periodic chain system, in Fig.[5.8]. Here we can also observe the discontinuity in concurrences for the different qubit pairs for the magnetic field of 0.96 units. The reason is the same as we had discussed earlier for the open-chain system only difference is in the transition point, which is 0.96 units for this case. In Fig.[5.8b] we have also provided the zoomed view of this plot Fig.[5.8a] to see the transition point clearly.



(a) Variation of Concurrences with magnetic field for four-qubit periodic chain Heisenberg-Fibonacci system.

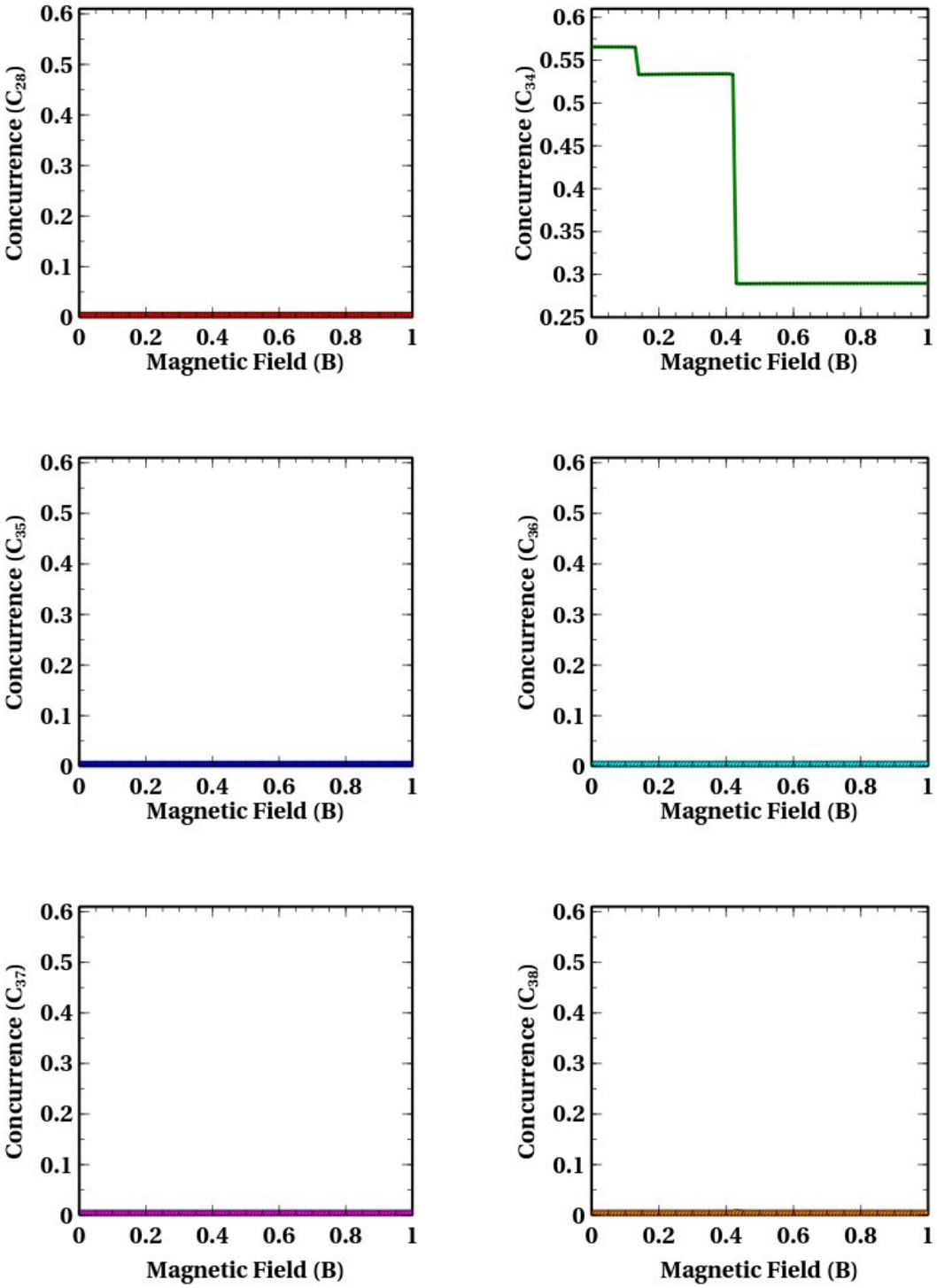


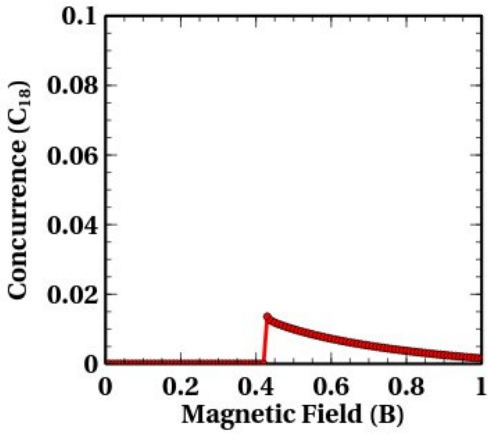
(b) Variation of Concurrences with magnetic field for four-qubit periodic chain Heisenberg-Fibonacci system (zoomed view).

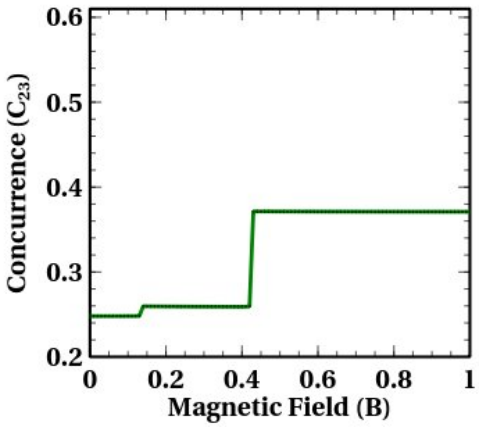
Figure 5.8: Variation of Concurrences with magnetic field for four-qubit periodic chain Heisenberg-like system.

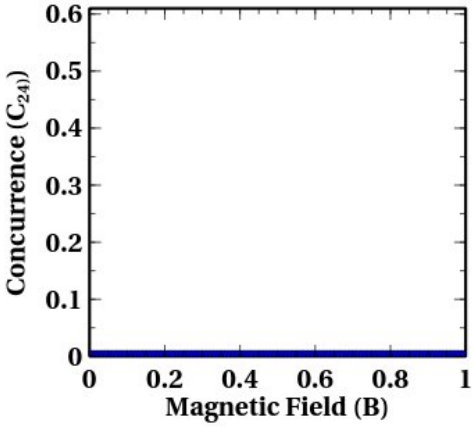
We can also see the variation of concurrences of various pairs for the 8-qubit periodic chain systems.

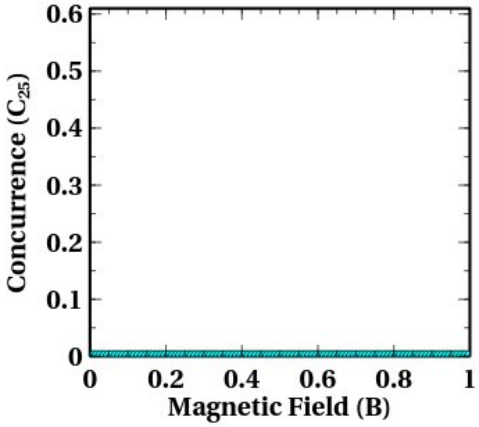
5.3.1.6 Concurrences for 8 qubit periodic Heisenberg-Fibonacci spin chain system

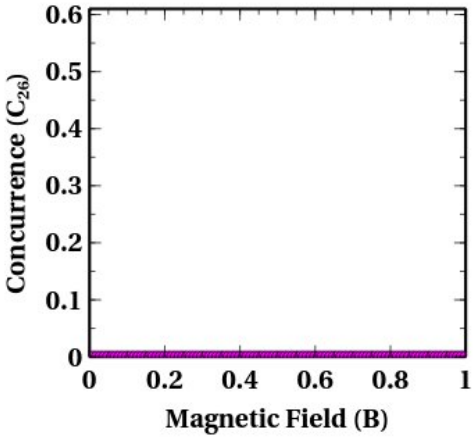


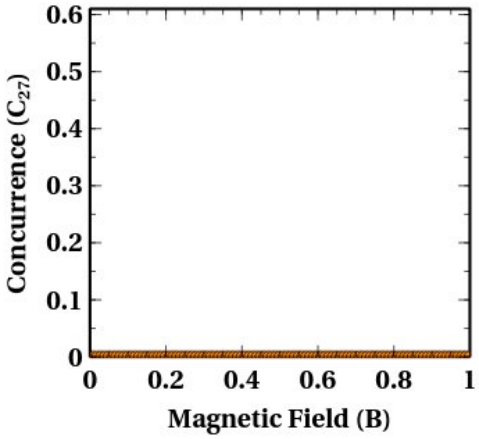

Concurrence (C_{18})
0.1
0.08
0.06
0.04
0.02
0
0
0.2
0.4
0.6
0.8
1
Magnetic Field (B)

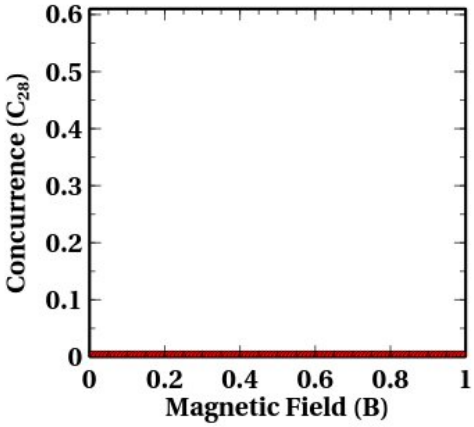

Concurrence (C_{23})
0.6
0.5
0.4
0.3
0.2
0
0.2
0.4
0.6
0.8
1
Magnetic Field (B)

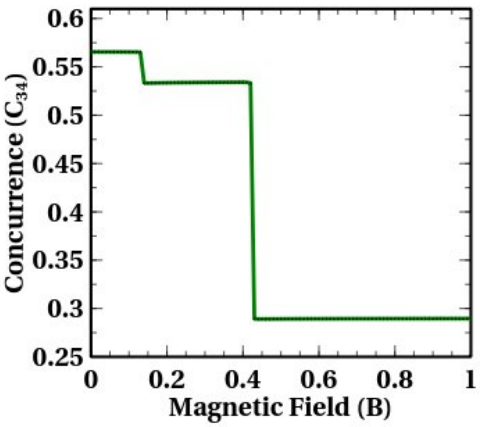

Concurrence (C_{24})
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1
Magnetic Field (B)

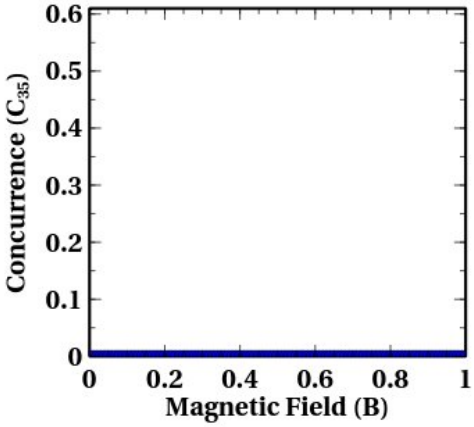

Concurrence (C_{25})
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1
Magnetic Field (B)

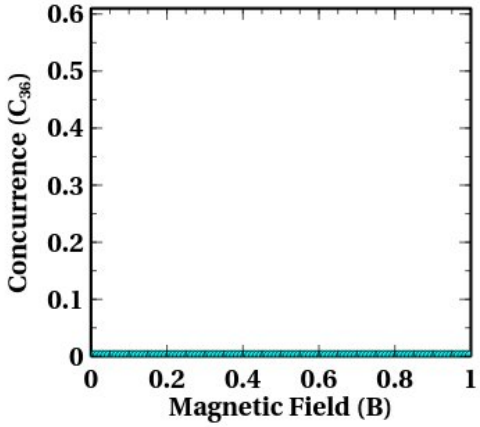

Concurrence (C_{26})
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1
Magnetic Field (B)

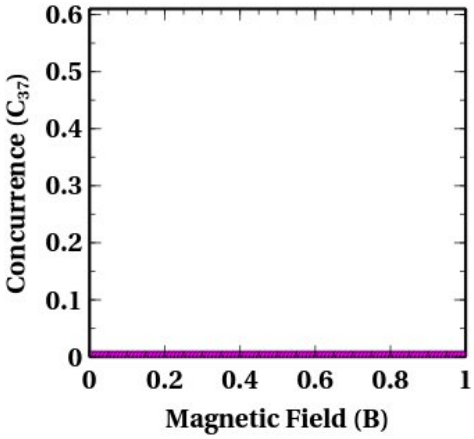

Concurrence (C_{27})
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1
Magnetic Field (B)

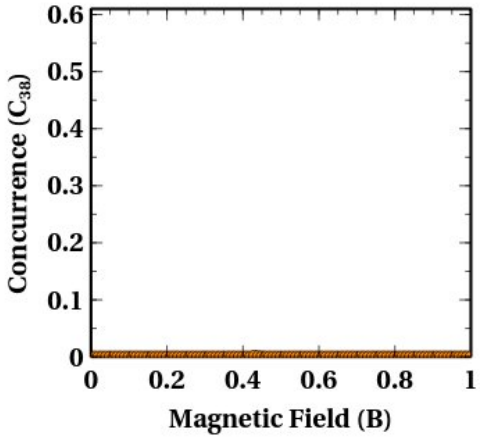

Concurrence (C_{28})
Magnetic Field (B)

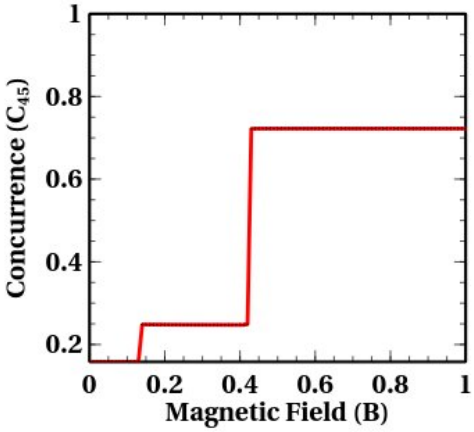

Concurrence (C_{34})
Magnetic Field (B)

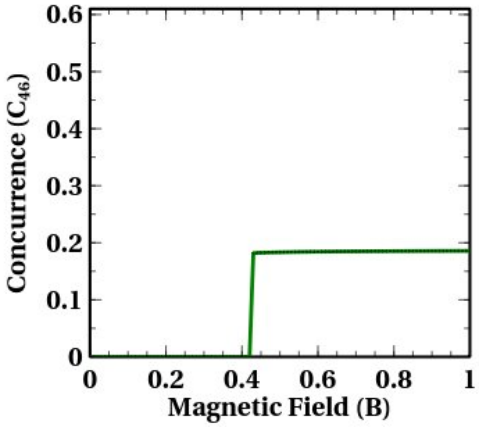

Concurrence (C_{35})
Magnetic Field (B)

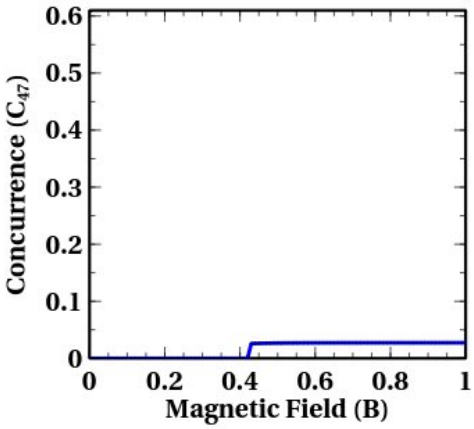

Concurrence (C_{36})
Magnetic Field (B)

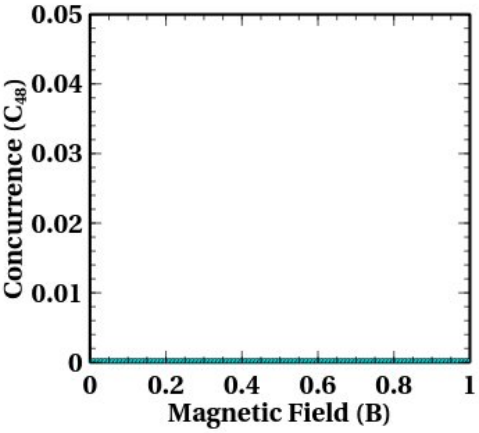

Concurrence (C_{37})
Magnetic Field (B)

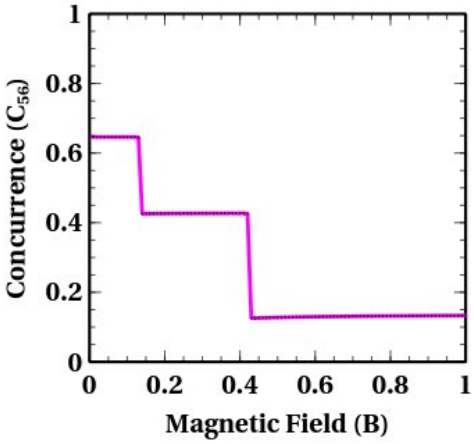

Concurrence (C_{38})
Magnetic Field (B)

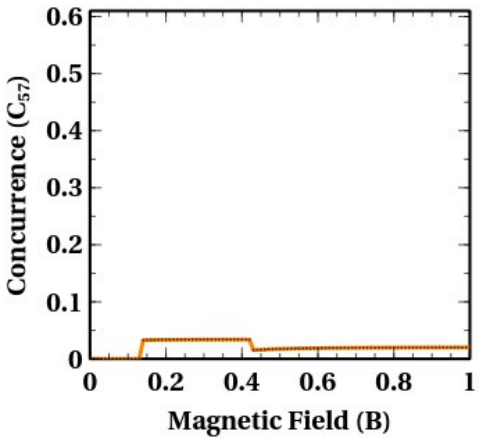

Concurrence (C_{45})
Magnetic Field (B)


Concurrence (C_{46})
Magnetic Field (B)


Concurrence (C_{47})
Magnetic Field (B)


Concurrence (C_{48})
Magnetic Field (B)


Concurrence (C_{56})
Magnetic Field (B)


Concurrence (C_{57})
Magnetic Field (B)

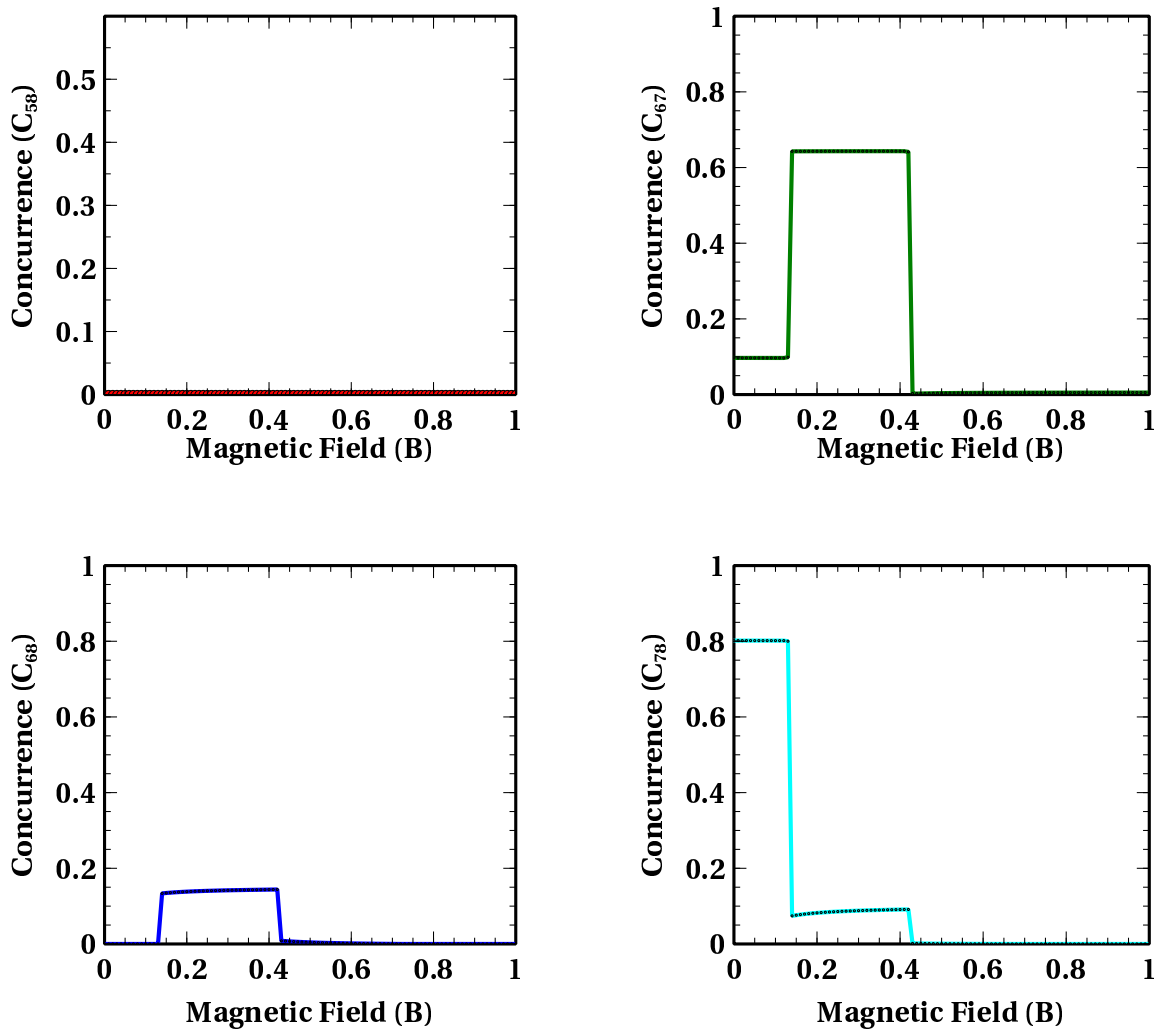


Figure 5.9: Variation of Concurrences with the Magnetic Field corresponding to 8-qubit open chain Heisenberg-like system.

5.3.2 Studies on von-Neumann Entropy

5.3.2.1 Case(1): Open Chain System

Let us now come to the von Neumann entropy of the system in the open chain configuration.

```
1 program Heisenberg_model_with_fibonacci_open_Entropy
2 !For n qubit open chain system
3 !Program of generalised Ising Model when J is varied like inverse of
      Fibonacci series and B is also varied
```

```
implicit none
integer :: i, n, num, o, a, x, y, k,  differing_bits
integer, parameter :: q = 1, t = 1, c = 1
integer, parameter :: d = 3
real*8 :: B
real*8 ::  be1
integer :: l
character(len=:), allocatable :: state(:)
integer, allocatable :: decimal(:)
integer, allocatable :: psi(:)
double precision, allocatable :: p(:), vec1(:), res1(:), vit(:)
integer :: j
real*8, allocatable :: S1(:,:), S2(:,:),S3(:,:),S4(:,:), S5(:,:), H
   (:,:), H1(:,:), rho(:,:)
integer, allocatable :: site(:)

! Parameters for LAPACK DSYEVX
integer :: LDA, info, LWORK, LDZ, M
real*8, allocatable :: Z(:,:)
integer, parameter :: IL = 1 , IU = 2
real*8, allocatable :: w(:), work(:)
character*1 :: JOBZ, UPLO
double precision :: VL, VU
integer, allocatable :: IFAIL(:), IWORK(:)
double precision, parameter :: ABSTOL = 0.0d0
character*1 :: RANGE   !'A' for computing all eigenvalues and 'I'
   for IL-th through IU-th eigenvalue

! Input values
print*, "Enter the number of bits you want:"
read*, n
num = 2**n ! Calculate number of states
l = n
! Allocate arrays after determining the value of num
allocate(character(len=n) :: state(num))
allocate(integer :: decimal(num))
allocate(psi(n))
allocate(S1(num, num))
```

```
allocate(S2(num, num))
allocate(S3(num, num))
allocate(S4(num, num))
allocate(S5(num, num))
allocate(H(num, num))
allocate(H1(num, num))
allocate(p(m))
allocate(vec1(num))
allocate(vit(0:num-1))
allocate(res1(num))
allocate(rho(num,num))
allocate(site(0:d-1))

! LAPACK parameters
LDA = num
LWORK = 8 * num + 1
M = IU-IL+1
LDZ = num
allocate(real(8) ::  w(LDA), work(LWORK))
allocate(integer :: IWORK(5*num), IFAIL(num))
allocate(real(8) :: Z(num,num))

! Printing the possible states
print*, "The states are:"
do i = 0, num - 1
    call integer_binary(i, state(i + 1), n)  ! Adjusting index for
   1-based
        print *, state(i + 1)
end do


! Convert binary states to decimal
    do i = 1, num
        decimal(i) = btod(state(i), n)
        !write(722,*) decimal(i)
    end do

call FibonacciSeries(l, p)
```

```
77
78 !Perform calculation based on sigma_i^x*sigma_i+1^x as S1
79 ! Initialize the S1 matrix
80         S1 = 0.0
81         do i = 1, num
82             do j = 1, num
83                 ! Initialize differing_bits counter
84                 differing_bits = 0
85                 do k = 0, n-1
86                     ! Compare the k-th bit of the i-th and j-th
    states
87                     if (btest(decimal(i), k) .neqv. btest(decimal(j)
    , k)) then
88                         differing_bits = differing_bits + 1
89                     end if
90                 end do
91                 ! Check if exactly two bits differ, and those bits
    are the specified positioned two qubits
92                 if (differing_bits == 2) then
93                     do o = 1,(n-1),1
94                         !print*, i, j, decimal(i), decimal(j)
95                         if ((btest(decimal(i), (n-o)) .neqv. btest(
    decimal(j), (n-o))) .and. &
96                             (btest(decimal(i), (n-o-1)) .neqv. btest
    (decimal(j), (n-o-1)))) then
97                             S1(i, j) = p(o)
98                             !print*, i, j, o, btest(decimal(i), (n-o
    )), btest(decimal(j), (n-o)), p(o), S1(i,j)
99                         end if
100                    end do
101                else
102                    S1(i, j) = 0.0
103                end if
104
105            end do
106        end do
107
108 !Perform calculation based on sigma_i^y*sigma_i+1^y as S2
```

```
109 ! Initialize the S2 matrix
110 S2 = 0.0
111     do i = 1, num
112         do j = 1, num
113         call single_digit(state(i), psi)
114         ! Initialize differing_bits counter
115         differing_bits = 0
116         do k = 0, n-1
117         ! Compare the k-th bit of the i-th and j-th states
118         if (btest(decimal(i), k) .neqv. btest(decimal(j), k)) then
119         differing_bits = differing_bits + 1
120         end if
121     end do
122     ! Check if exactly two bits differ, and those bits are the
    specified positioned two qubits
123     if (differing_bits == 2) then
124         do o = 1,(n-1),1
125         !print*, i, j, decimal(i), decimal(j)
126             if ((btest(decimal(i), (n-o)) .neqv. btest(decimal(j), (
    n-o))) .and. &
127             (btest(decimal(i), (n-o-1)) .neqv. btest(decimal(j), (n-
    o-1)))) then
128             S2(i, j) = (-1)**(psi(o) + psi(o+1) + 1)*p(o)
129             !print*, i, j, o, btest(decimal(i), (n-o)), btest(
    decimal(j), (n-o)), p(o), S1(i,j)
130             end if
131         end do
132             else
133                 S2(i, j) = 0.0
134             end if
135             end do
136         end do
137
138 !Perform calculation based on sigma_i^z*sigma_i+1^z as S3
139 ! Perform calculations based on the decimal values for S3
140 do i = 1, num
141     do j = 1, num
142         call single_digit(state(i), psi)
```

```
143         if (decimal(i) /= decimal(j)) then
144             S3(i, j) = 0.0
145         else
146         S3(i, j) = 0.0
147         do k = 2, n
148             S3(i, j) = S3(i, j) + p(k-1)*(1.0 - 2.0 * psi(k-1)) *
    (1.0 - 2.0 * psi(k))
149         end do
150         end if
151     end do
152 end do
153
154 ! Construct the S4 matrix When magnetic field is along x direction (
    for sigma_x )
155 S4 = 0.0
156 do i = 1, num
157     do j = 1, num
158     ! Initialize differing_bits counter
159         differing_bits = 0
160         do k = 1, n
161             ! Compare the k-th bit of the i-th and j-th states
162             if (btest(decimal(i), k-1) .neqv. btest(decimal(j), k-1)
    ) then
163                 differing_bits = differing_bits + 1
164             end if
165         end do
166         ! If exactly one bit differs, set S4(i, j) = 1.0
167         if (differing_bits == 1) then
168             S4(i, j) = 1.0
169         else
170             S4(i, j) = 0.0
171         end if
172     end do
173 end do
174
175 !constructing the matrix S5 When the magnetic field is along the z
    direction (for sigma_z )
176 do i = 1, num
```

```
177     do j = 1, num
178     call single_digit(state(i), psi)
179     if (decimal(i) /= decimal(j)) then
180         S5(i, j) = 0.0
181     else
182         S5(i, j) = 0.0
183     do k = 1, n
184         S5(i, j) = S5(i,j) + (1.0 - 2.0 * psi(k))
185     end do
186     end if
187     end do
188 end do
189
190 ! Total Hamiltonian
191 do y = 0, 100, 1
192     B = y * 0.01
193     H =  (q*S1 + t*S2 + c*S3 ) + B * S4
194     ! Ground State and 1st excited state Eigenvalue and eigenvector
    determination using DSYEVX
195     JOBZ = 'V'
196     UPLO = 'U'
197     RANGE = 'I'
198     call DSYEVX (JOBZ, RANGE, UPLO, num, H, LDA, VL, VU, IL, IU,
    ABSTOL, M, W, Z, LDZ, WORK, LWORK, IWORK, IFAIL, INFO)
199     vec1 = Z(:, 1)
200     do i = 1, num
201         vit(i-1) = vec1(i)
202     end do
203     write(2,*) vit
204     if (info == 0) then
205         do a = 1, LDA
206             write(17,*) B, w(a) !plot for magfield vs eigenvalue
207         end do
208     else
209         write(15,*) 'Error in DSYEV, INFO=', info
210     end if
211
212   !Von Neumann Entropy
```

```
213     do i = 0, n-2
214         do j = i+1, n-1, 1
215             site(0) = i
216             site(1) = j
217             call BERPVR(n,2,site,vit,be1)
218             write(841,*) b, be1
219         end do
220     end do
221 end do
222 
223 deallocate(state)
224 deallocate(decimal)
225 deallocate(psi)
226 deallocate(S1)
227 deallocate(S2)
228 deallocate(S3)
229 deallocate(S4)
230 deallocate(S5)
231 deallocate(H)
232 deallocate(H1)
233 deallocate(p)
234 deallocate(rho)
235 deallocate(vit)
236 deallocate(vec1)
237 deallocate(site)
238 
239 contains
240 function btod(state, n) result(decimal)
241     implicit none
242     integer, intent(in) :: n
243     character(len=n) :: state
244     integer :: decimal, i
245     decimal = 0
246     do i = 1, n
247         if (state(i:i) == '1') then
248             decimal = decimal + 2**(n-i)
249         end if
250     end do
```

```
251 end function btod
252
253     subroutine integer_binary(num, binary_string, n)
254         implicit none
255         integer :: num
256         integer, intent(in) :: n
257         integer :: i, temp_num
258         character(len=n), intent(out) :: binary_string
259         temp_num = num
260         binary_string = repeat('0', n)
261         do i = 0, n - 1
262             if (mod(temp_num, 2) == 1) then
263                 binary_string(n-i:n-i) = '1'
264             else
265                 binary_string(n-i:n-i) = '0'
266             end if
267             temp_num = temp_num / 2
268         end do
269     end subroutine integer_binary
270
271     subroutine single_digit(A, psi)
272         implicit none
273         character(len=*), intent(in) :: A
274         integer, intent(out) :: psi(:)
275         integer :: i
276         do i = 1, len(A)
277             if (A(i:i) == '0') then
278                 psi(i) = 0
279             else if (A(i:i) == '1') then
280                 psi(i) = 1
281             else
282                 print *, "Invalid input in single_digit subroutine"
283             end if
284         end do
285     end subroutine single_digit
286
287     subroutine FibonacciSeries(m, reciprocal)
288         implicit none
```

```
289         integer :: i
290         integer :: m
291         double precision :: fib(m)
292         double precision :: reciprocal(m)
293         if (m <= 0) then
294             print*, "Number of terms must be greater than 0."
295             stop
296         end if
297         ! Initialize the first two terms
298         fib(1) = 1
299         if (n > 1) then
300             fib(2) = 1
301         end if
302         ! Generate the Fibonacci series
303         do i = 3, m
304             fib(i) = fib(i-1) + fib(i-2)
305         end do
306         ! Compute reciprocals
307         do i = 1, m
308             reciprocal(i) = 1.0d0 / real(fib(i))
309         end do
310         !print*, fib
311     end subroutine FibonacciSeries
312 subroutine BERPVR(s,s1,site,vin,be)
313     implicit none
314     integer::s, s1,site(0:s1-1)
315     real*8::be,betemp,vin(0:2**s-1),rdm(0:2**s1-1,0:2**s1-1)
316     integer::i,INFO
317     real*8::WORK(0:3*(2**s1)-1),W(0:2**s1-1)
318     call PTRVR(s,s1,site,vin,rdm)
319     call DSYEV('N','U', 2**s1, rdm, 2**s1, &
320     W, WORK,3*(2**s1)-1,INFO)
321     be=0
322     betemp=0
323     do i=0,2**s1-1,1
324         if (W(i)>0.0000000000001) then
325         betemp=-(dlog(abs(W(i)))*abs(W(i)))/dlog(2.0d0)
326         be=be+betemp
```

```
        end if
    end do
end subroutine

    subroutine PTRVR(s, s1, site, vin, rdm)
        implicit none
        integer :: s, s1, s2
        real*8 :: trace
        real*8, dimension(0:2**s-1) :: vin
        real*8, dimension(0:2**s1-1, 0:2**s1-1) :: rdm
        integer :: ii, i1, a, i0, ia, j1, oo, k1, x1, y1, j, t1, t2,
    z1
        integer, dimension(0:s1-1) :: site, site2
        integer, dimension(0:s1-1) :: bin
        integer, dimension(0:(2**(s-s1))*(2**s1)-1) :: ind
        s2 = s - s1
        x1 = 0
        y1 = 0
        ind = 0
        do j1 = 2**s1-1, 0, -1
            do ii = 0, 2**s-1
                a = ii
                do i1 = 0, s1-1
                    i0 = site(i1)
                    call DTOBONEBIT(a, ia, i0, s)
                    site2(i1) = ia
                end do
                call DTOB(j1, bin, s1)
                oo = 1
                do k1 = 0, s1-1
                    oo = oo * (bin(k1) - site2(k1))
                end do
                if (abs(oo) == 1) then
                    ind(x1) = ii
                    x1 = x1 + 1
                end if
            end do
        end do
```

```
364
365             rdm = 0.0d0
366             do t1 = 0, 2**s1-1
367                 do t2 = 0, 2**s1-1
368                     do z1 = 0, 2**s2-1
369                         rdm(t1, t2) = rdm(t1, t2) + vin(ind((2**s2)
    * t1 + z1)) * vin(ind((2**s2) * t2 + z1))
370                     end do
371                 end do
372             end do
373         end subroutine
374
375         subroutine DTOBONEBIT(m, ia, i0, s)
376             implicit none
377             integer*4 :: s
378             integer :: tt(0:s-1)
379             integer :: m, ia, i0
380             call DTOB(m, tt, s)
381             ia = tt(i0)
382         end subroutine
383
384         subroutine DTOB(m, tt, s)
385             implicit none
386             integer :: s
387             integer, dimension(0:s-1) :: tt
388             integer :: m, k, a2
389             tt = 0
390             a2 = m
391             do k = 0, s-1
392                 tt(s-k-1) = mod(a2, 2)
393                 a2 = a2 / 2
394                 if (a2 == 0) exit
395             end do
396         end subroutine
397
398 end program Heisenberg_model_with_fibonacci_open_entropy
```

We can see that the variation of the von Neumann entropy corresponding to the block consisting of (1,2), (2,3), and (1,3) is constant with the change in magnetic field for the three-qubit open chain system, as is evident from the Fig.[5.10]. It is also noticeable that, since the entanglement between the qubit pair (1,2) is stronger, it makes the block consisting of the qubit pair (1,2) less entangled with the other block, containing only one of the qubits.

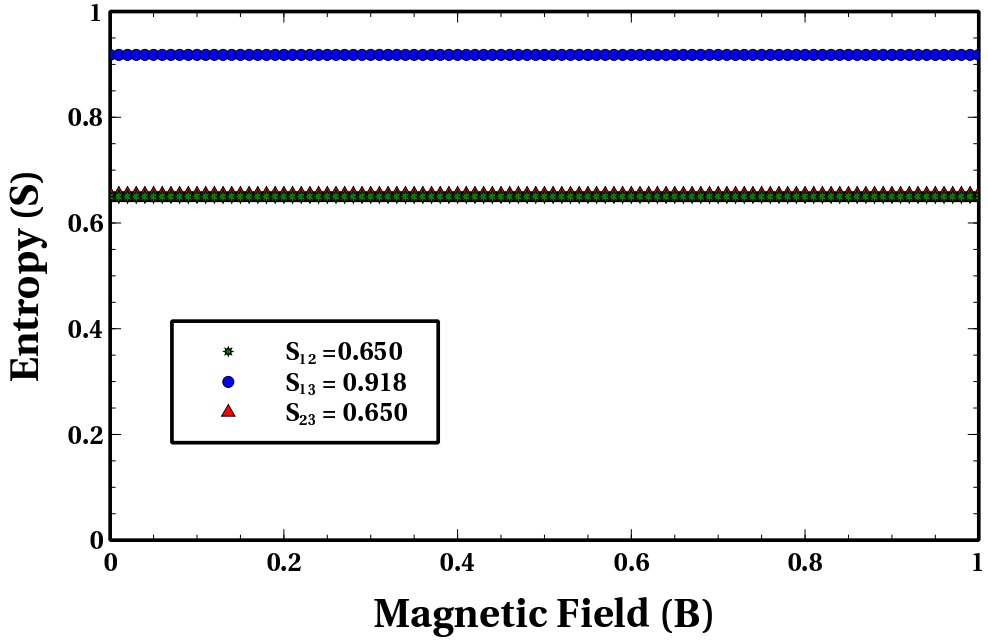


Figure 5.10: Variation of von Neumann entropy with magnetic field for three-qubit open chain Heisenberg-Fibonacci system.

Let us see the von Neumann entropy for the 4-qubit open chain system. For that purpose, we can give the value of $n = 4$ in the above program. We can see from the Fig.[5.11] below how the entropy corresponding to different blocks consisting of the qubit pairs (1,2), (1,3), (1,4),(2,3), (2,4) & (3,4) vary with the change in magnetic field.

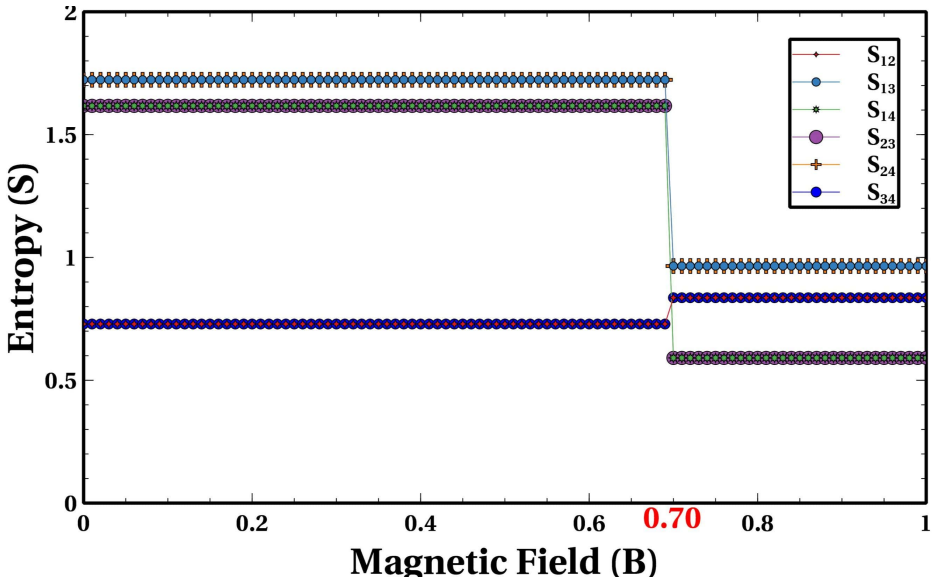
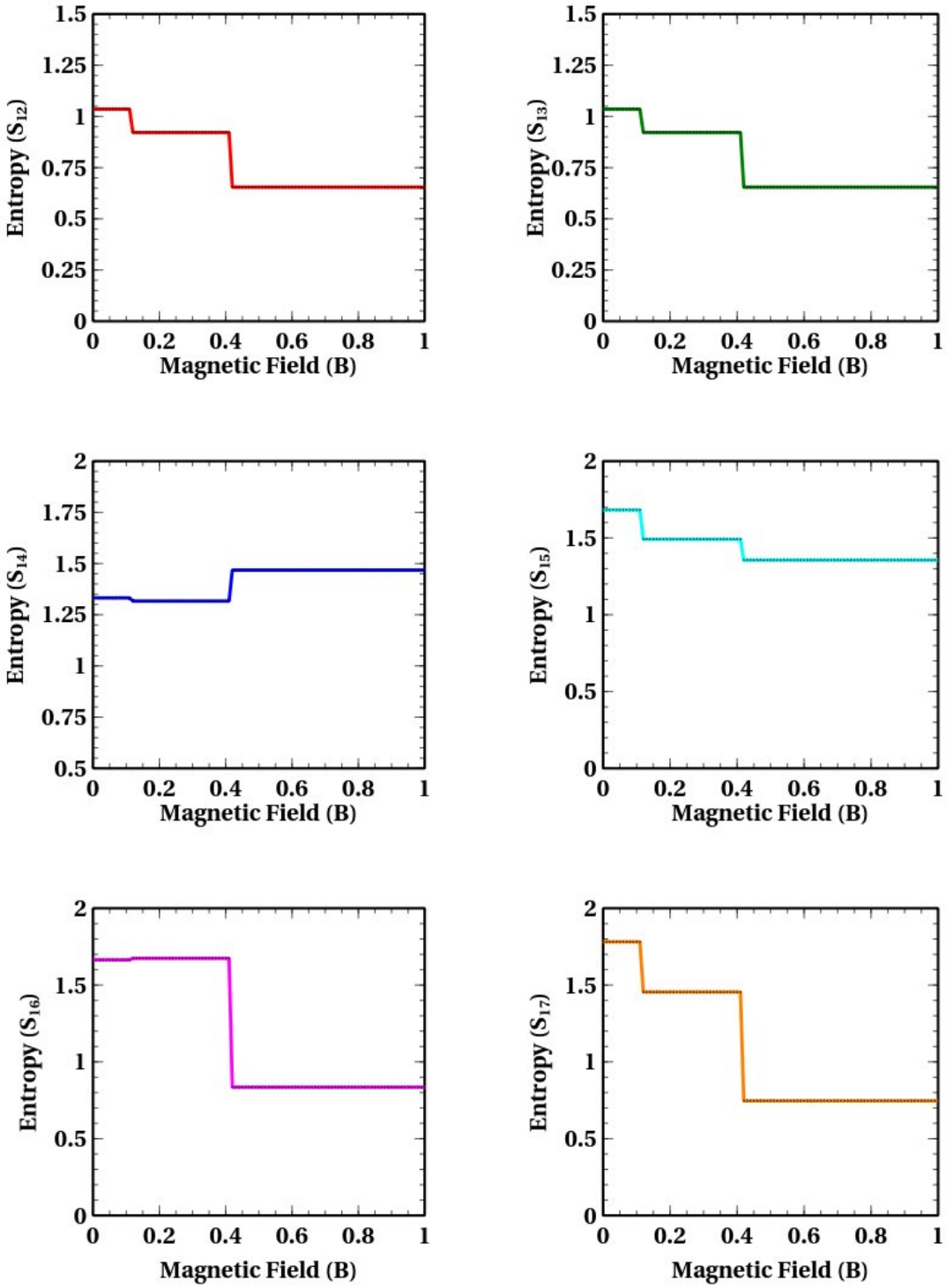
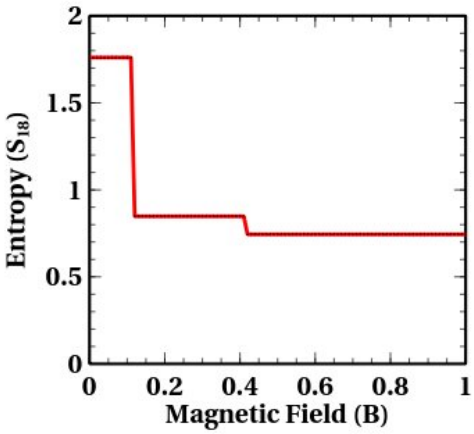


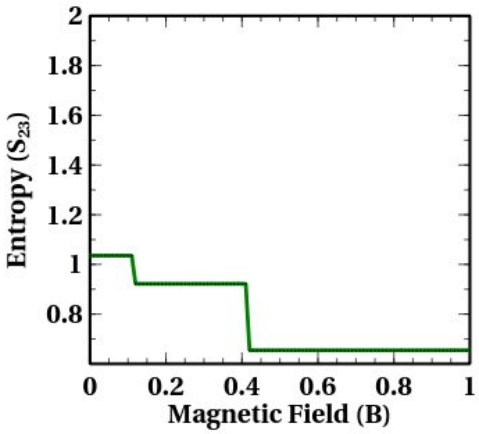
Figure 5.11: Variation of von Neumann entropy with magnetic field for four-qubit open chain Heisenberg-Fibonacci system.

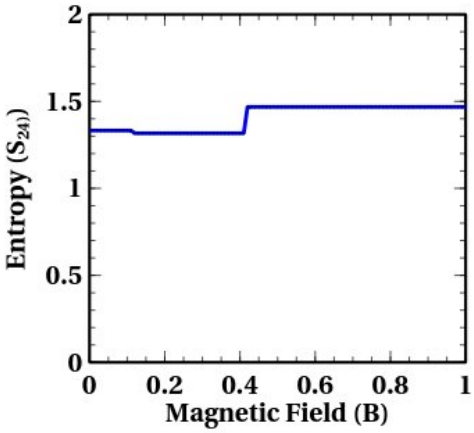
Here we are also observing the discontinuity in the variation of the entropies, signifying the quantum phase transition of the system about the magnetic field of 0.70 unit. The reason for this has already been stated. We can also see the variation of the von Neumann entropies for an 8-qubit open chain system. In this 8-qubit open chain system also we can also see that the block entropies corresponding to the block consisting of two consecutive qubits are higher at the beginning, and they decrease with the increase in the magnetic field. This fact indicates that the entanglement between consecutive qubit pairs is higher, which opposes the qubit pair from undergoing entanglement with any other qubits.

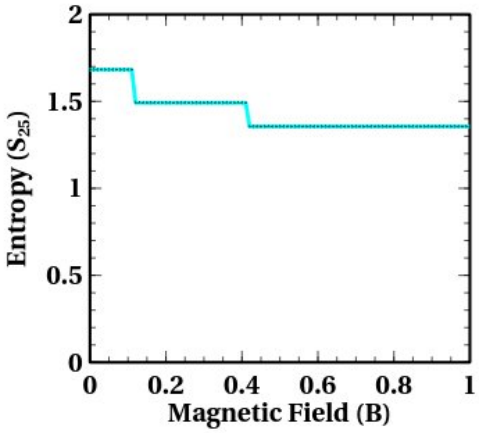
5.3.2.2 Block entropies for 8 qubit open Heisenberg-Fibonacci spin chain system

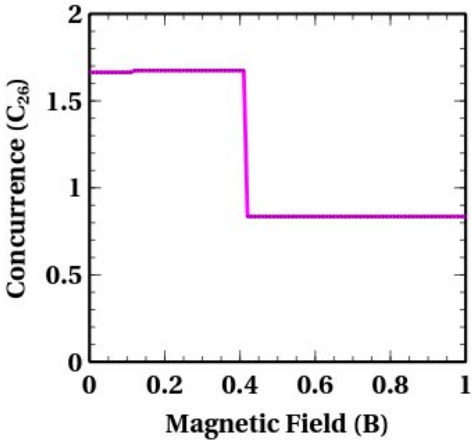


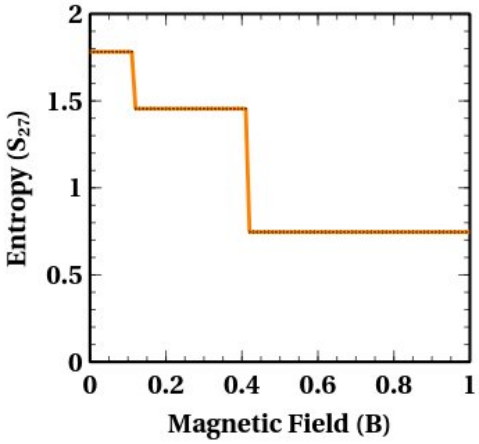

Entropy (S_{18})
Magnetic Field (B)

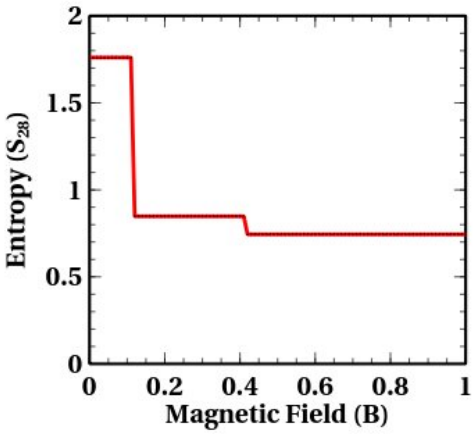

Entropy (S_{23})
Magnetic Field (B)

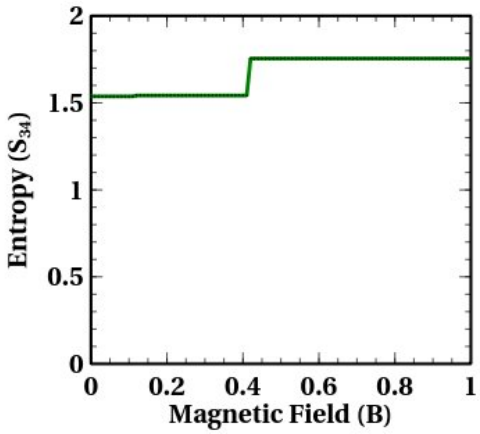

Entropy (S_{24})
Magnetic Field (B)

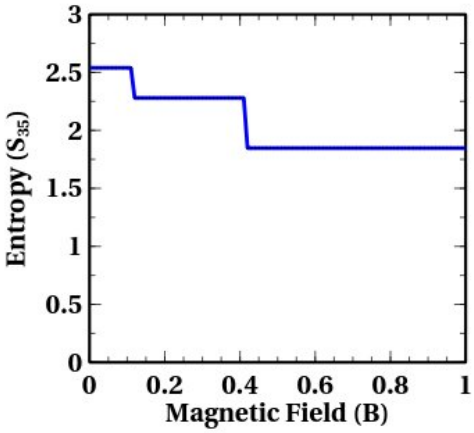

Entropy (S_{25})
Magnetic Field (B)

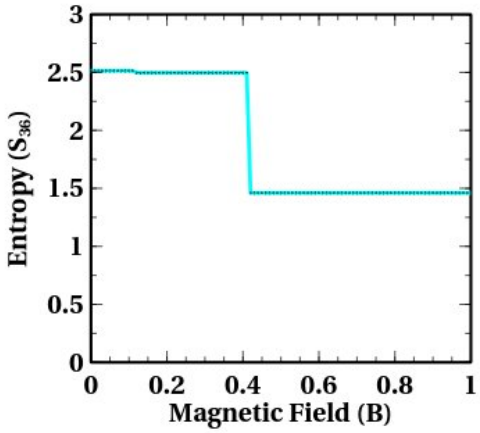

Concurrence (C_{26})
Magnetic Field (B)

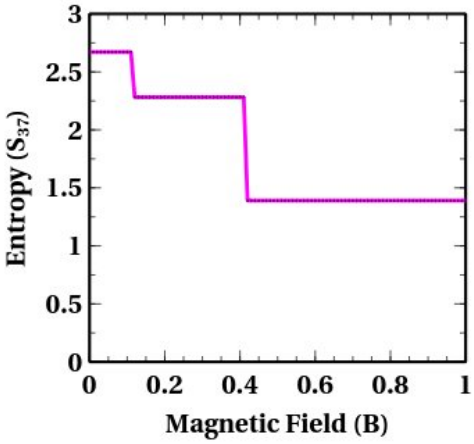

Entropy (S_{27})
Magnetic Field (B)

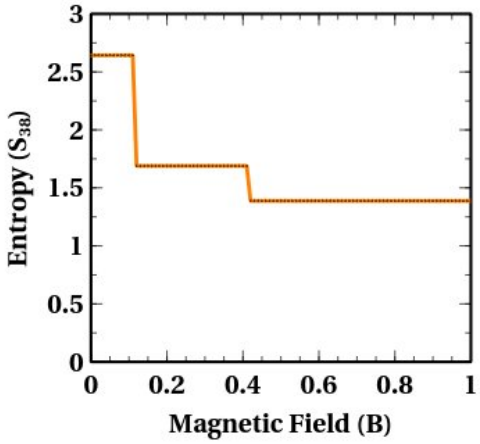

Entropy (S_{28})
Magnetic Field (B)
0
0.5
1
1.5
2
0.2
0.4
0.6
0.8

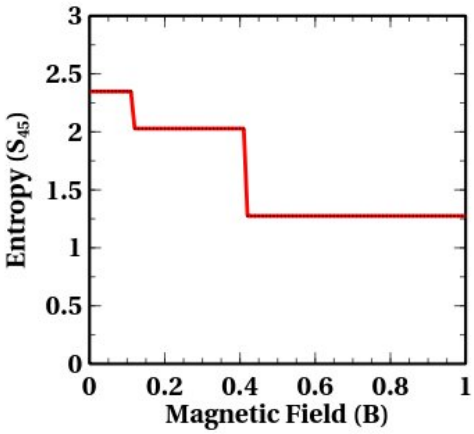

Entropy (S_{34})
Magnetic Field (B)
0
0.5
1
1.5
2
0.2
0.4
0.6
0.8

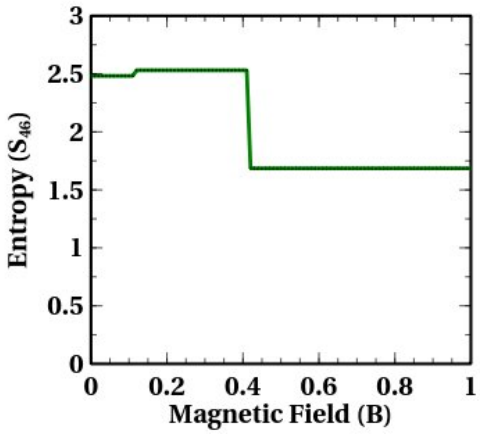

Entropy (S_{35})
Magnetic Field (B)
0
0.5
1
1.5
2
2.5
3
0.2
0.4
0.6
0.8

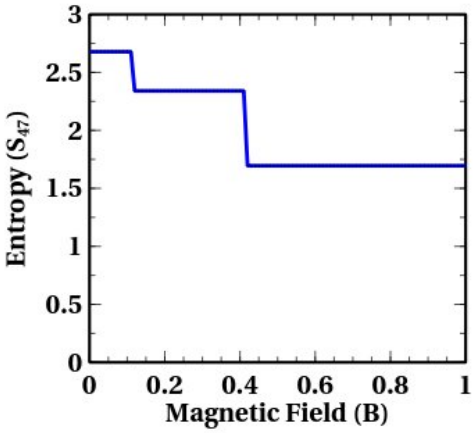

Entropy (S_{36})
Magnetic Field (B)
0
0.5
1
1.5
2
2.5
3
0.2
0.4
0.6
0.8

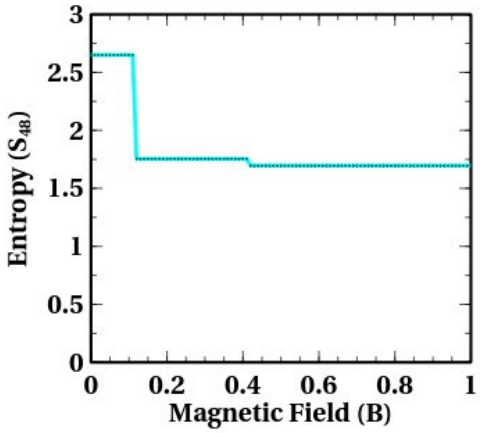

Entropy (S_{37})
Magnetic Field (B)
0
0.5
1
1.5
2
2.5
3
0.2
0.4
0.6
0.8

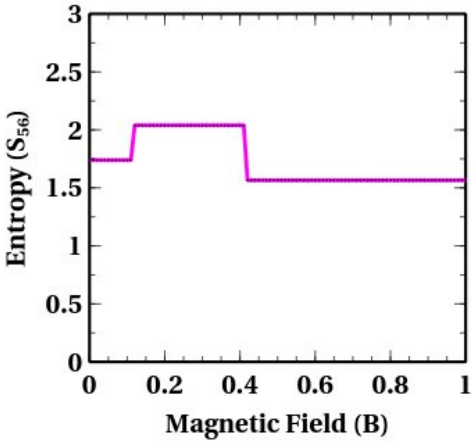

Entropy (S_{38})
Magnetic Field (B)
0
0.5
1
1.5
2
2.5
3
0.2
0.4
0.6
0.8

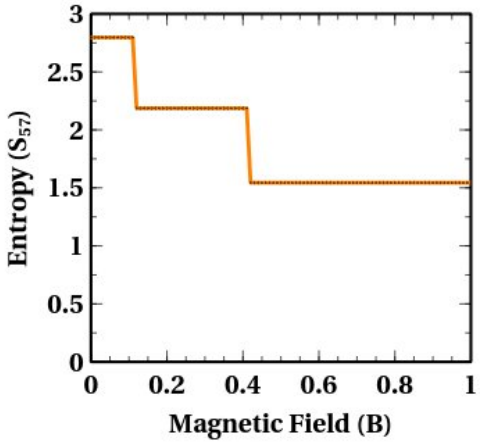

Entropy (S_{45})
Magnetic Field (B)
0
0.5
1
1.5
2
2.5
3
0.2
0.4
0.6
0.8

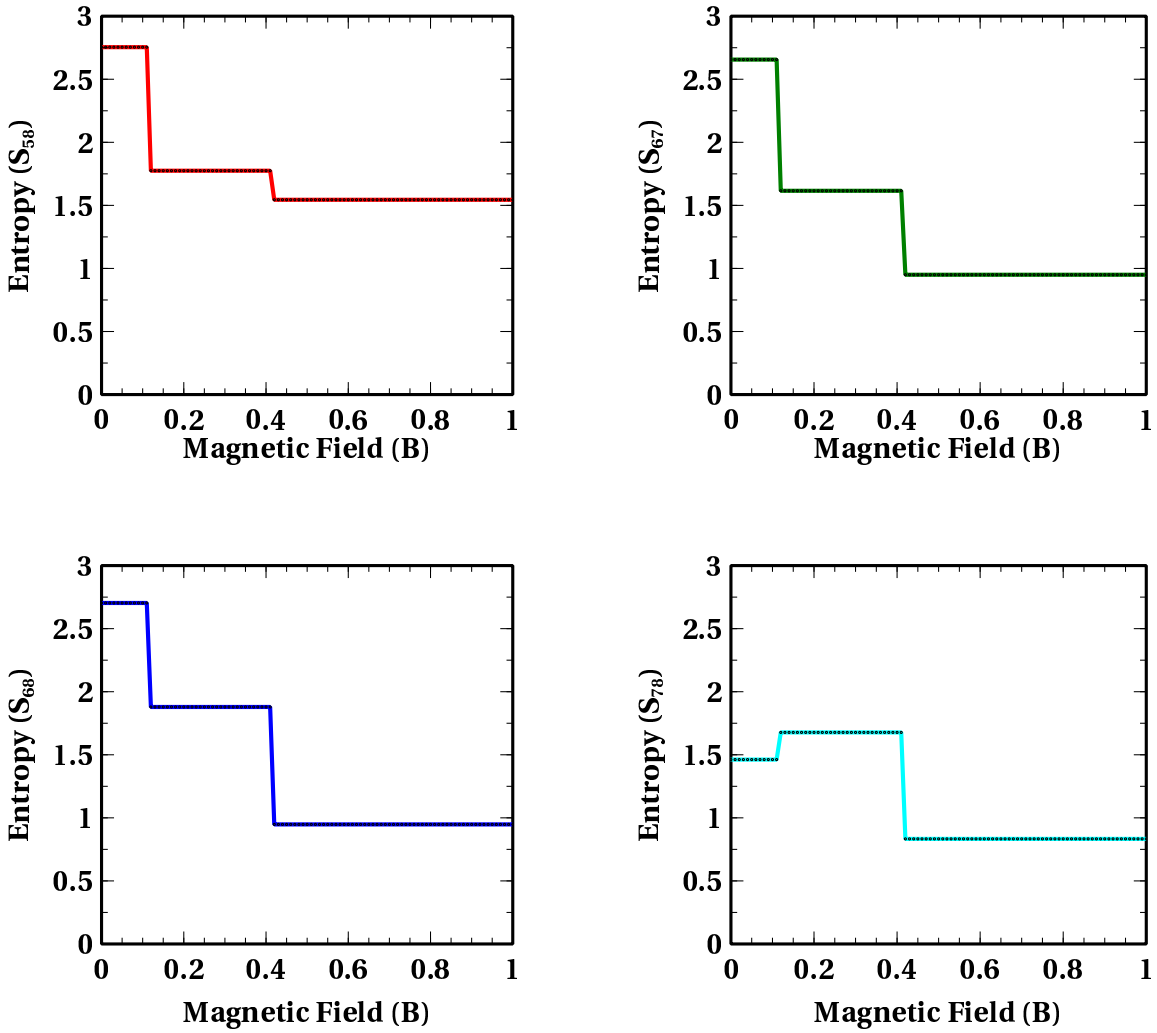

Entropy (S_{46})
Magnetic Field (B)
0
0.5
1
1.5
2
2.5
3
0.2
0.4
0.6
0.8


Entropy (S_{47})
Magnetic Field (B)
0
0.5
1
1.5
2
2.5
3
0.2
0.4
0.6
0.8


Entropy (S_{48})
Magnetic Field (B)
0
0.5
1
1.5
2
2.5
3
0.2
0.4
0.6
0.8


Entropy (S_{56})
Magnetic Field (B)
0
0.5
1
1.5
2
2.5
3
0.2
0.4
0.6
0.8


Entropy (S_{57})
Magnetic Field (B)
0
0.5
1
1.5
2
2.5
3
0.2
0.4
0.6
0.8



5.3.2.3 Case(2): Periodic Chain System

Now we can write the following Fortran code to study the von Neumann entropies for periodic chain systems. For this case, we have to utilize the wrap-up condition as $\sigma_{n+1} \equiv sigma_1$.

```
program Heisenberg_model_with_fibonacci_Periodic_Entropy
!For n qubit periodic chain system
!Program of generalised Ising Model when J is varied like inverse of
    Fibonacci series and B is also varied
implicit none
integer :: i, n, num, o, a, x, y, k,  differing_bits
integer, parameter :: q = 1, t = 1, c = 1
```

```
integer, parameter :: d = 3
real*8 :: B
real :: con, be2, con1
real*8 ::  be1
integer :: l
character(len=:), allocatable :: state(:)
integer, allocatable :: decimal(:)
integer, allocatable :: psi(:)
double precision, allocatable :: p(:), vec1(:), res1(:), vit(:)
integer :: j
real*8, allocatable :: S1(:,:), S2(:,:),S3(:,:),S4(:,:), S5(:,:), H
   (:,:), H1(:,:), rho(:,:)
integer, allocatable :: site(:)

! Parameters for LAPACK DSYEVX
integer :: LDA, info, LWORK, LDZ, M
real*8, allocatable :: Z(:,:)
integer, parameter :: IL = 1 , IU = 2
real*8, allocatable :: w(:), work(:)
character*1 :: JOBZ, UPLO
double precision :: VL, VU
integer, allocatable :: IFAIL(:), IWORK(:)
double precision, parameter :: ABSTOL = 0.0d0
character*1 :: RANGE   !'A' for computing all eigenvalues and 'I'
   for IL-th through IU-th eigenvalue

! Input values
print*, "Enter the number of bits you want:"
read*, n
num = 2**n ! Calculate number of states
l = n
! Allocate arrays after determining the value of num
allocate(character(len=n) :: state(num))
allocate(integer :: decimal(num))
allocate(psi(n))
allocate(S1(num, num))
allocate(S2(num, num))
allocate(S3(num, num))
```

```
allocate(S4(num, num))
allocate(S5(num, num))
allocate(H(num, num))
allocate(H1(num, num))
allocate(p(m))
allocate(vec1(num))
allocate(vit(0:num-1))
allocate(res1(num))
allocate(rho(num,num))
allocate(site(0:d-1))

! LAPACK parameters
LDA = num
LWORK = 8 * num + 1
M = IU-IL+1
LDZ = num
allocate(real(8) ::  w(LDA), work(LWORK))
allocate(integer :: IWORK(5*num), IFAIL(num))
allocate(real(8) :: Z(num,num))

! Printing the possible states
print*, "The states are:"
do i = 0, num - 1
    call integer_binary(i, state(i + 1), n)  ! Adjusting index for
   1-based
        print *, state(i + 1)
end do


! Convert binary states to decimal
    do i = 1, num
        decimal(i) = btod(state(i), n)
        !write(722,*) decimal(i)
    end do

call FibonacciSeries(l, p)

!Perform calculation based on sigma_i^x*sigma_i+1^x as S1
```

```
! Initialize the S1 matrix
      S1 = 0.0
      do i = 1, num
          do j = 1, num
              ! Initialize differing_bits counter
              differing_bits = 0
              do k = 0, n-1
                  ! Compare the k-th bit of the i-th and j-th states
                  if (btest(decimal(i), k) .neqv. btest(decimal(j), k)) then
                      differing_bits = differing_bits + 1
                  end if
              end do
              ! Check if exactly two bits differ, and those bits are the specified positioned two qubits
              if (differing_bits == 2) then
                  do o = 1,(n-1),1
                      !print*, i, j, decimal(i), decimal(j)
                      if ((btest(decimal(i), (n-o)) .neqv. btest(decimal(j), (n-o))) .and. &
                          (btest(decimal(i), (n-o-1)) .neqv. btest(decimal(j), (n-o-1)))) then
                          S1(i, j) = p(o)
                          !print*, i, j, o, btest(decimal(i), (n-o)), btest(decimal(j), (n-o)), p(o), S1(i,j)
                      end if
                  end do
              else
                  S1(i, j) = 0.0
              end if

          end do
      end do

!Perform calculation based on sigma_i^y*sigma_i+1^y as S2
! Initialize the S2 matrix
S2 = 0.0
```

```
112 do i = 1, num
113     do j = 1, num
114         call single_digit(state(i), psi)
115         ! Initialize differing_bits counter
116         differing_bits = 0
117         do k = 0, n-1
118         ! Compare the k-th bit of the i-th and j-th states
119         if (btest(decimal(i), k) .neqv. btest(decimal(j), k)) then
120         differing_bits = differing_bits + 1
121         end if
122         end do
123     ! Check if exactly two bits differ, and those bits are the
    specified positioned two qubits
124     if (differing_bits == 2) then
125         do o = 1,(n-1),1
126         if ((btest(decimal(i), (n-o)) .neqv. btest(decimal(j), (n-o)
    )) .and. &
127         (btest(decimal(i), (n-o-1)) .neqv. btest(decimal(j), (n-o-1)
    ))) then
128             S2(i, j) = (-1)**(psi(o) + psi(o+1) + 1)*p(o)
129             !print*, i, j, o, btest(decimal(i), (n-o)), btest(
    decimal(j), (n-o)), p(o), S1(i,j)
130         end if
131         end do
132         if ((btest(decimal(i), (n-1)) .neqv. btest(decimal(j), (n-1)
    )) .and. &
133         (btest(decimal(i), 0) .neqv. btest(decimal(j), 0))) then
134         S2(i, j) = (-1)**(psi(n-2) + psi(n) + 1)*p(n)
135         end if
136         else
137             S2(i, j) = 0.0
138         end if
139     end do
140 end do
141 !Perform calculation based on sigma_i^z*sigma_i+1^z as S3
142 ! Perform calculations based on the decimal values for S3
143      do i = 1, num
144         do j = 1, num
```

```
145             call single_digit(state(i), psi)
146             if (decimal(i) /= decimal(j)) then
147                 S3(i, j) = 0.0
148             else
149                 S3(i, j) = 0.0
150                 do k = 1, n - 1
151                     S3(i, j) = S3(i, j) + p(k)*(1.0 - 2.0 * psi(k))
    * (1.0 - 2.0 * psi(k + 1))
152                 end do
153                 S3(i, j) = S3(i, j) + p(n)*(1.0 - 2.0 * psi(n)) *
    (1.0 - 2.0 * psi(1))
154             end if
155         end do
156     end do
157
158     ! Construct the S4 matrix (for sigma_x )
159     S4 = 0.0
160     do i = 1, num
161         do j = 1, num
162             ! Initialize differing_bits counter
163             differing_bits = 0
164             do k = 1, n
165                 ! Compare the k-th bit of the i-th and j-th states
166                 if (btest(decimal(i), k-1) .neqv. btest(decimal(j),
    k-1)) then
167                     differing_bits = differing_bits + 1
168                 end if
169             end do
170             ! If exactly one bit differs, set S4(i, j) = 1.0
171             if (differing_bits == 1) then
172                 S4(i, j) = 1.0
173             else
174                 S4(i, j) = 0.0
175             end if
176         end do
177     end do
178
179 ! Constructing the matrix S5 When the magnetic field is along the z
```

```
    direction (for sigma_z )
do i = 1, num
    do j = 1, num
    call single_digit(state(i), psi)
    if (decimal(i) /= decimal(j)) then
        S5(i, j) = 0.0
    else
        S5(i, j) = 0.0
    do k = 1, n
        S5(i, j) = S5(i,j) + (1.0 - 2.0 * psi(k))
    end do
    end if
    end do
end do

! Total Hamiltonian
do y = 0, 100, 1
    B = y * 0.01
    H =  (q*S1 + t*S2 + c*S3 ) + B * S4
    ! Ground State and 1st excited state Eigenvalue and eigenvector
    determination using DSYEVX
    JOBZ = 'V'
    UPLO = 'U'
    RANGE = 'I'
    call DSYEVX (JOBZ, RANGE, UPLO, num, H, LDA, VL, VU, IL, IU,
    ABSTOL, M, W, Z, LDZ, WORK, LWORK, IWORK, IFAIL, INFO)
    vec1 = Z(:, 1)
    do i = 1, num
        vit(i-1) = vec1(i)
    end do
    write(2,*) vit
    if (info == 0) then
        do a = 1, LDA
            write(17,*) B, w(a) !plot for magfield vs eigenvalue
        end do
    else
        write(15,*) 'Error in DSYEV, INFO=', info
    end if
```

```
215
216     !Concurrence
217     do i = 0, n-1
218         do j = i+1, n-1
219             if( i /= (n-1)) then
220             call CONVR(n,i,j,vit,con1)
221             write(8001,*) B , con1
222             else
223                 EXIT
224             end if
225         end do
226     end do
227 end do
228
229 deallocate(state)
230 deallocate(decimal)
231 deallocate(psi)
232 deallocate(S1)
233 deallocate(S2)
234 deallocate(S3)
235 deallocate(S4)
236 deallocate(S5)
237 deallocate(H)
238 deallocate(H1)
239 deallocate(p)
240 deallocate(rho)
241 deallocate(vit)
242 deallocate(vec1)
243 deallocate(site)
244
245 contains
246 function btod(state, n) result(decimal)
247     implicit none
248     integer, intent(in) :: n
249     character(len=n) :: state
250     integer :: decimal, i
251     decimal = 0
252     do i = 1, n
```

```
253         if (state(i:i) == '1') then
254             decimal = decimal + 2**(n-i)
255         end if
256     end do
257 end function btod
258 
259     subroutine integer_binary(num, binary_string, n)
260         implicit none
261         integer :: num
262         integer, intent(in) :: n
263         integer :: i, temp_num
264         character(len=n), intent(out) :: binary_string
265         temp_num = num
266         binary_string = repeat('0', n)
267         do i = 0, n - 1
268             if (mod(temp_num, 2) == 1) then
269                 binary_string(n-i:n-i) = '1'
270             else
271                 binary_string(n-i:n-i) = '0'
272             end if
273             temp_num = temp_num / 2
274         end do
275     end subroutine integer_binary
276 
277     subroutine single_digit(A, psi)
278         implicit none
279         character(len=*), intent(in) :: A
280         integer, intent(out) :: psi(:)
281         integer :: i
282         do i = 1, len(A)
283             if (A(i:i) == '0') then
284                 psi(i) = 0
285             else if (A(i:i) == '1') then
286                 psi(i) = 1
287             else
288                 print *, "Invalid input in single_digit subroutine"
289             end if
290         end do
```

```
291     end subroutine single_digit
292
293     subroutine FibonacciSeries(m, reciprocal)
294         implicit none
295         integer :: i
296         integer :: m
297         double precision :: fib(m)
298         double precision :: reciprocal(m)
299         if (m <= 0) then
300             print*, "Number of terms must be greater than 0."
301             stop
302         end if
303         ! Initialize the first two terms
304         fib(1) = 1
305         if (n > 1) then
306             fib(2) = 1
307         end if
308         ! Generate the Fibonacci series
309         do i = 3, m
310             fib(i) = fib(i-1) + fib(i-2)
311         end do
312         ! Compute reciprocals
313         do i = 1, m
314             reciprocal(i) = 1.0d0 / real(fib(i))
315         end do
316         !print*, fib
317     end subroutine FibonacciSeries
318 subroutine BERPVR(s,s1,site,vin,be)
319     implicit none
320     integer::s, s1,site(0:s1-1)
321     real*8::be,betemp,vin(0:2**s-1),rdm(0:2**s1-1,0:2**s1-1)
322     integer::i,INFO
323     real*8::WORK(0:3*(2**s1)-1),W(0:2**s1-1)
324     call PTRVR(s,s1,site,vin,rdm)
325     call DSYEV('N','U', 2**s1, rdm, 2**s1, &
326     W, WORK,3*(2**s1)-1,INFO)
327     be=0
328     betemp=0
```

```
329     do i=0,2**s1-1,1
330         if (W(i)>0.0000000000001) then
331         betemp=-(dlog(abs(W(i)))*abs(W(i)))/dlog(2.0d0)
332         be=be+betemp
333         end if
334     end do
335 end subroutine
336 subroutine PTRVR(s, s1, site, vin, rdm)
337     implicit none
338     integer :: s, s1, s2
339     real*8 :: trace
340     real*8, dimension(0:2**s-1) :: vin
341     real*8, dimension(0:2**s1-1, 0:2**s1-1) :: rdm
342     integer :: ii, i1, a, i0, ia, j1, oo, k1, x1, y1, j, t1, t2, z1
343     integer, dimension(0:s1-1) :: site, site2
344     integer, dimension(0:s1-1) :: bin
345     integer, dimension(0:(2**(s-s1))*(2**s1)-1) :: ind
346     s2 = s - s1
347     x1 = 0
348     y1 = 0
349     ind = 0
350     do j1 = 2**s1-1, 0, -1
351         do ii = 0, 2**s-1
352             a = ii
353             do i1 = 0, s1-1
354                 i0 = site(i1)
355                 call DTOBONEBIT(a, ia, i0, s)
356                 site2(i1) = ia
357             end do
358             call DTOB(j1, bin, s1)
359             oo = 1
360             do k1 = 0, s1-1
361                 oo = oo * (bin(k1) - site2(k1))
362             end do
363             if (abs(oo) == 1) then
364                 ind(x1) = ii
365                 x1 = x1 + 1
366             end if
```

```
367         end do
368     end do
369     rdm = 0.0d0
370     do t1 = 0, 2**s1-1
371         do t2 = 0, 2**s1-1
372             do z1 = 0, 2**s2-1
373                 rdm(t1, t2) = rdm(t1, t2) + vin(ind((2**s2) * t1 +
    z1)) * vin(ind((2**s2) * t2 + z1))
374             end do
375         end do
376     end do
377 end subroutine
378
379 subroutine DTOBONEBIT(m, ia, i0, s)
380 implicit none
381 integer*4 :: s
382 integer :: tt(0:s-1)
383 integer :: m, ia, i0
384 call DTOB(m, tt, s)
385 ia = tt(i0)
386 end subroutine
387
388 subroutine DTOB(m, tt, s)
389 implicit none
390 integer :: s
391 integer, dimension(0:s-1) :: tt
392 integer :: m, k, a2
393 tt = 0
394 a2 = m
395 do k = 0, s-1
396     tt(s-k-1) = mod(a2, 2)
397     a2 = a2 / 2
398     if (a2 == 0) exit
399     end do
400 end subroutine
401
402 end program Heisenberg_model_with_fibonacci_Periodic_Entropy
```

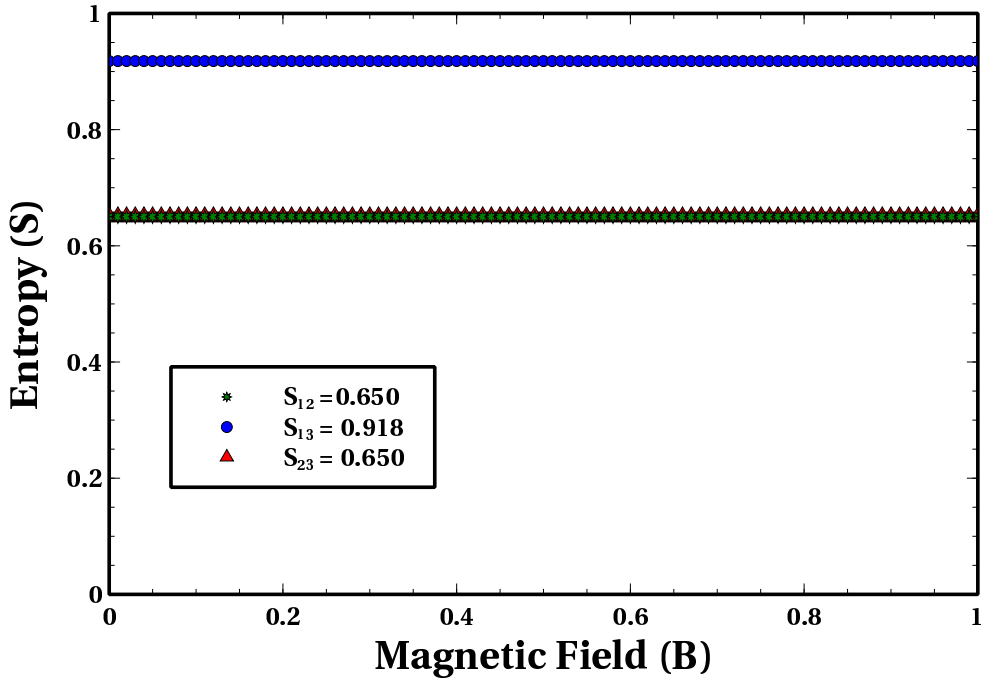
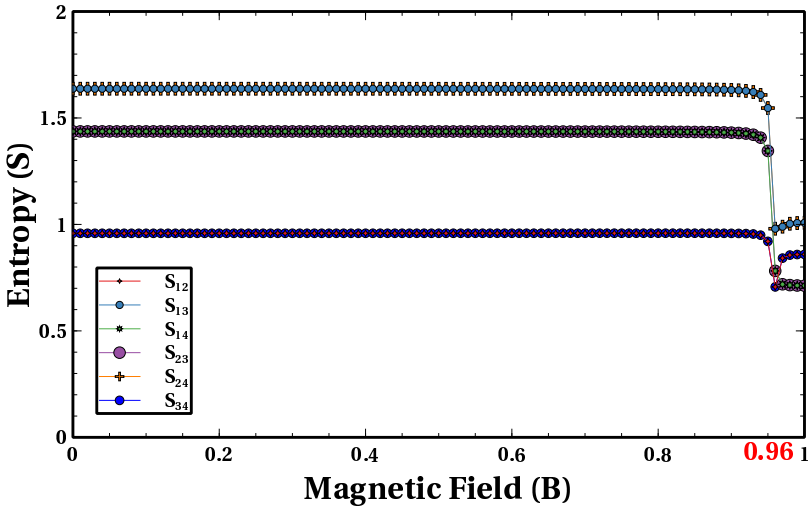
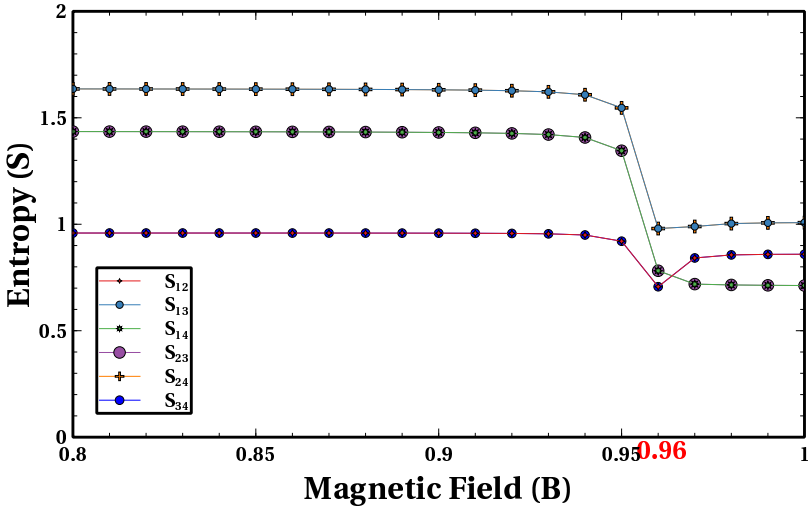


Figure 5.12: Variation of von Neumann entropy with magnetic field for three-qubit periodic chain Heisenberg-Fibonacci system.

It is observed that for the three-qubit periodic chain system, the von Neumann entropies are continuous and also constant with respect to the change in the magnetic field. Its value as calculated is $S_{12} = 0.650$,S_{13}=0.918, and $S_{23} = 0.650$, respectively. This indicates the larger entanglement between the blocks of qubit pairs (1,3) and the other block containing qubit 2. Let us go for a 4-qubit system. The variation of the von Neumann entropies is shown in the Fig.[5.13a]. We have observed a quantum phase transition at a magnetic field of 0.96 units. The zoomed portion is also given in Fig.[5.13b].



(a) Variation of von Neumann entropy with magnetic field for four-qubit periodic chain Heisenberg-Fibonacci system.

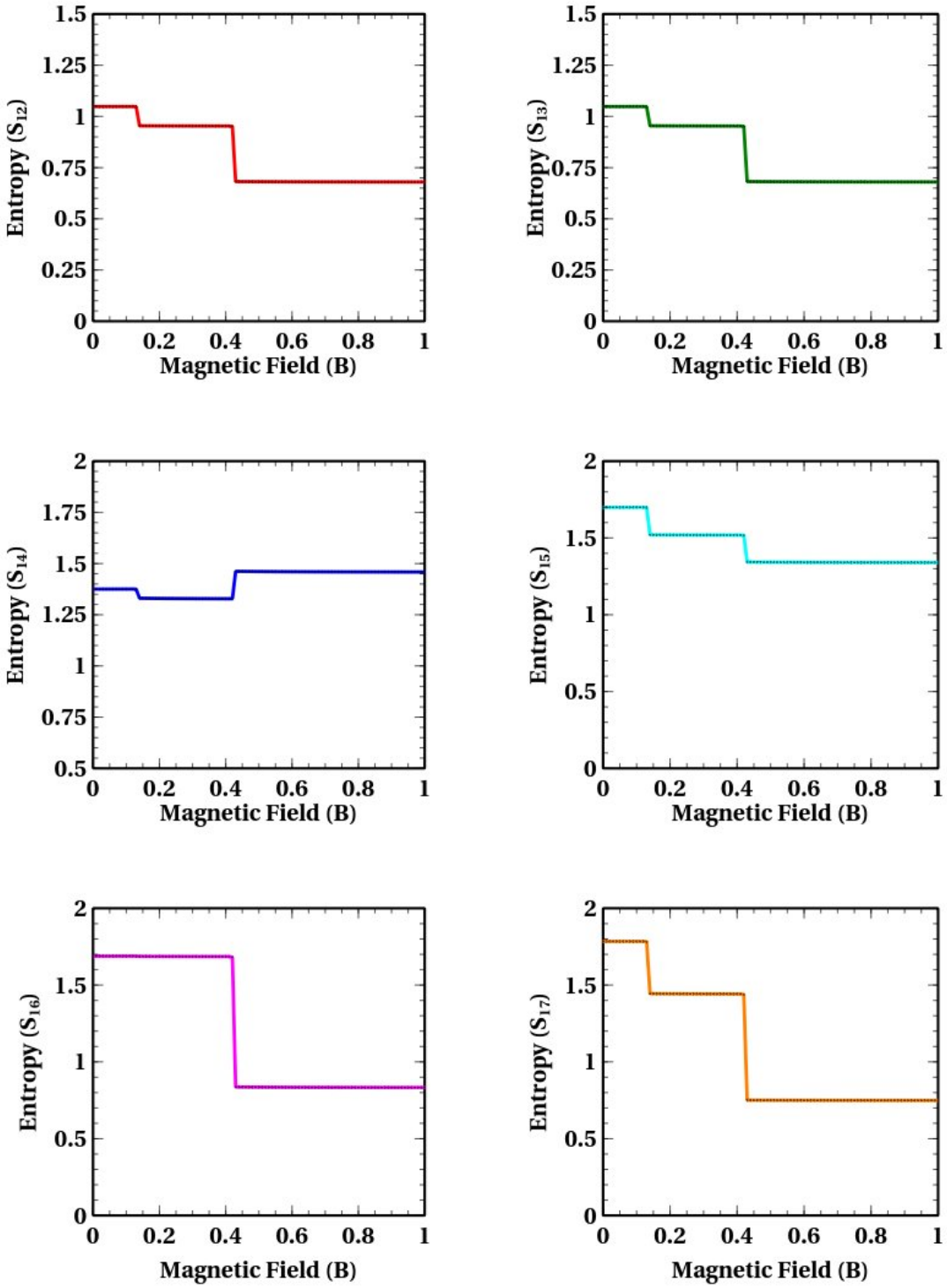


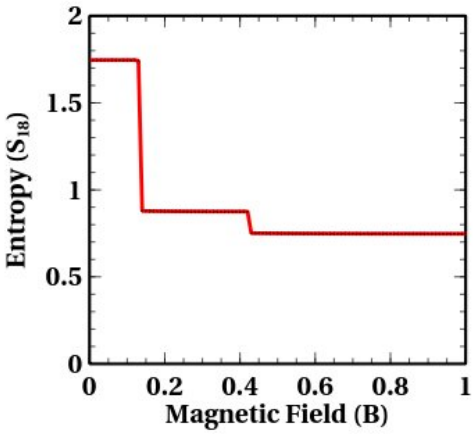
(b) Variation of von Neumann entropy with magnetic field for four-qubit periodic chain Heisenberg-Fibonacci system (zoomed view).

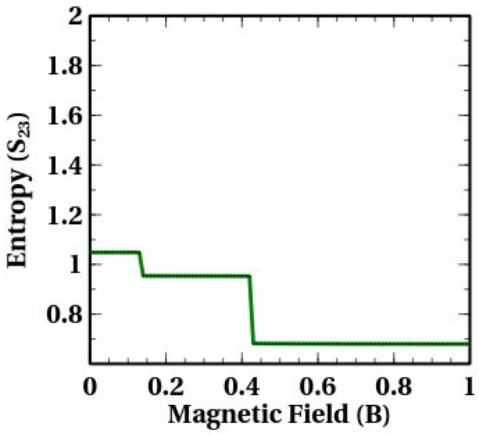
Figure 5.13: Variation of von Neumann entropy with magnetic field for four-qubit periodic chain Heisenberg-Fibonacci system.

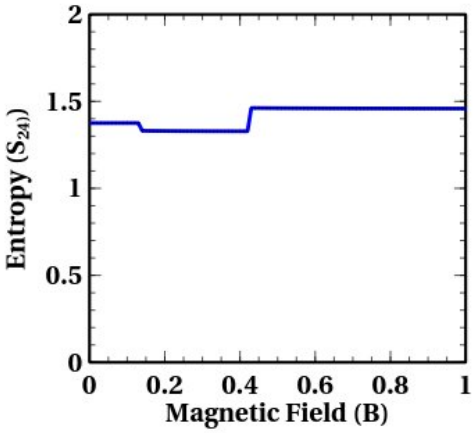
We can also see the variation of von Neumann entropies corresponding to an 8-qubit periodic chain system in Fig.[5.14] where we can see more than one transition point.

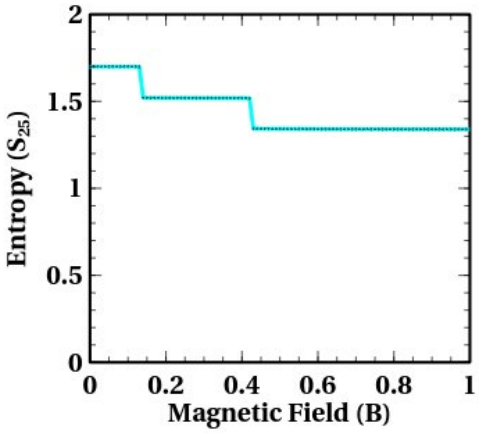
5.3.2.4 Block entropies for 8 qubit periodic Heisenberg-Fibonacci spin chain system

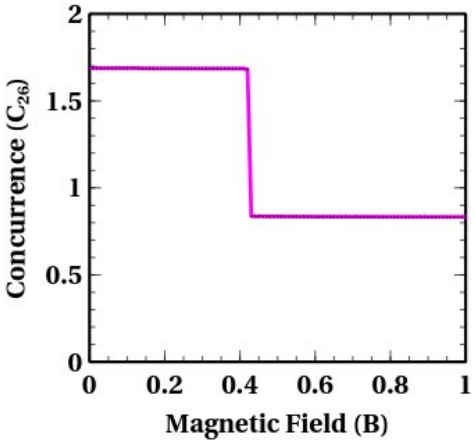


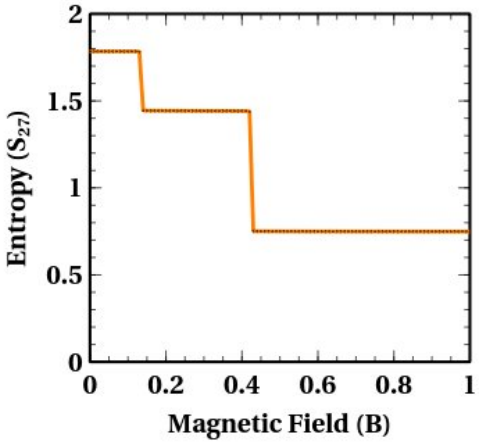

Entropy (S_{18})
Magnetic Field (B)

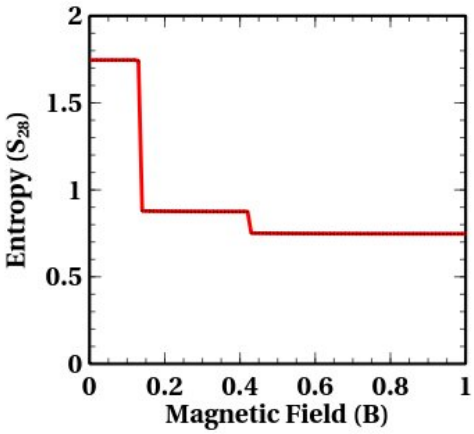

Entropy (S_{23})
Magnetic Field (B)

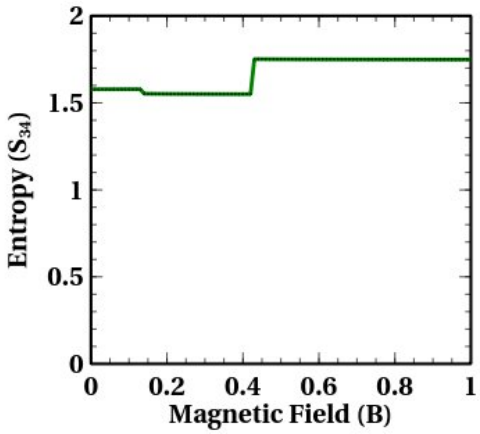

Entropy (S_{24})
Magnetic Field (B)

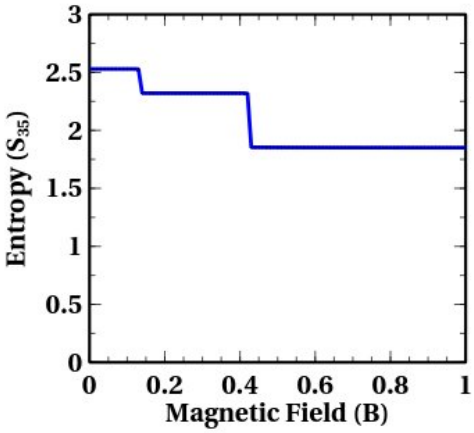

Entropy (S_{25})
Magnetic Field (B)

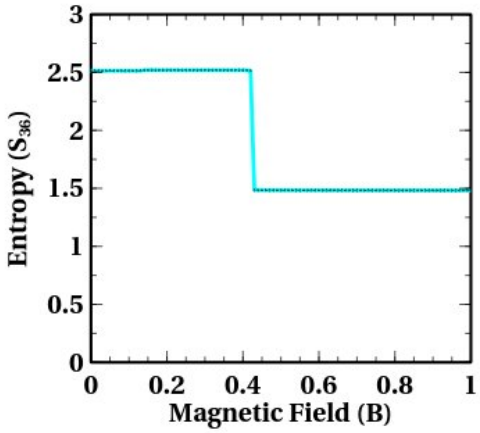

Concurrence (C_{26})
Magnetic Field (B)

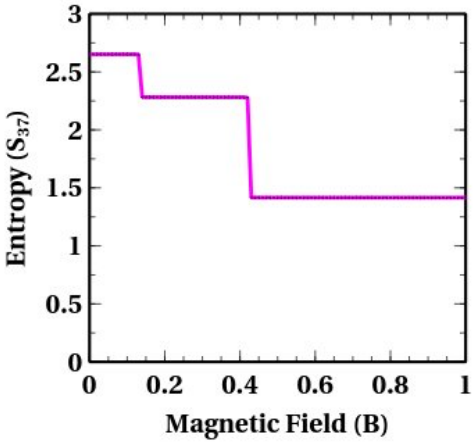

Entropy (S_{27})
Magnetic Field (B)

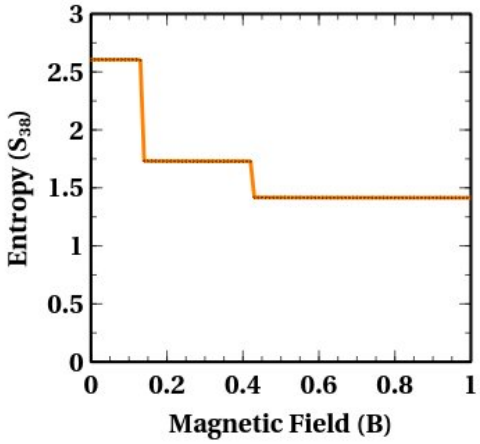

Entropy (S_{28})
Magnetic Field (B)
0
0.5
1
1.5
2
0.2
0.4
0.6
0.8

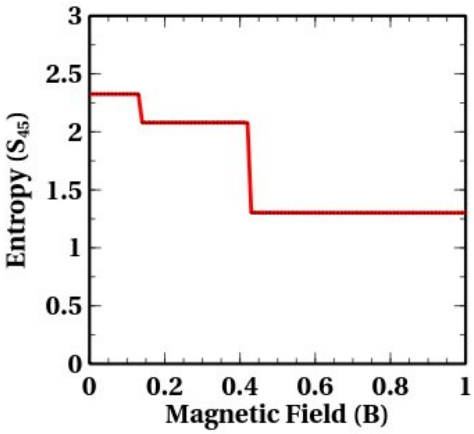

Entropy (S_{34})
Magnetic Field (B)
0
0.5
1
1.5
2
0.2
0.4
0.6
0.8

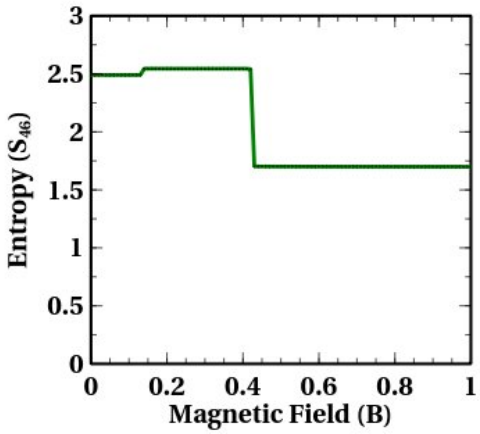

Entropy (S_{35})
Magnetic Field (B)
0
0.5
1
1.5
2
2.5
3
0.2
0.4
0.6
0.8

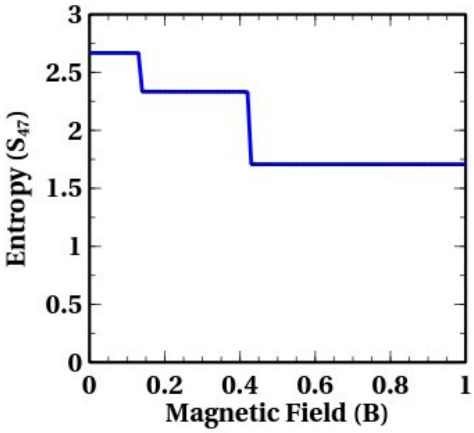

Entropy (S_{36})
Magnetic Field (B)
0
0.5
1
1.5
2
2.5
3
0.2
0.4
0.6
0.8

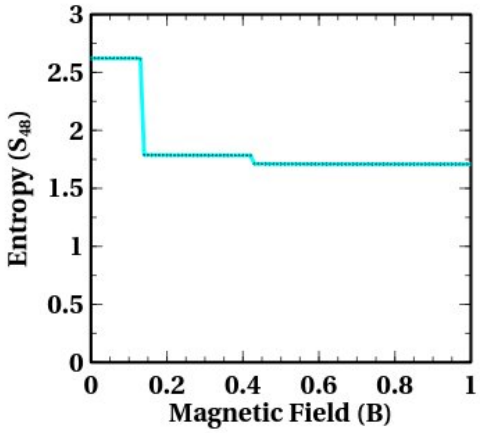

Entropy (S_{37})
Magnetic Field (B)
0
0.5
1
1.5
2
2.5
3
0.2
0.4
0.6
0.8

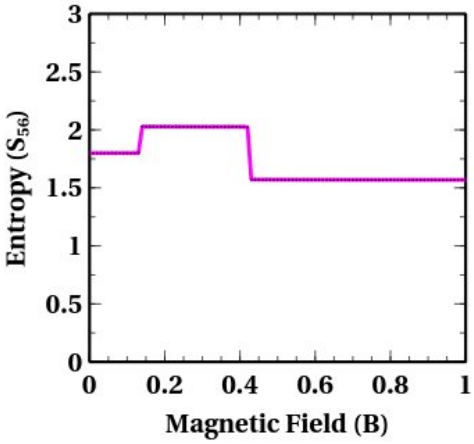

Entropy (S_{38})
Magnetic Field (B)
0
0.5
1
1.5
2
2.5
3
0.2
0.4
0.6
0.8

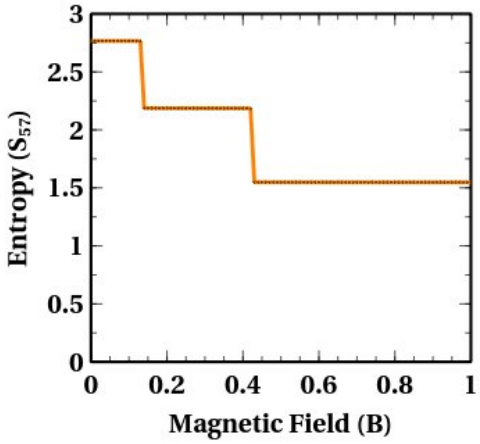

Entropy (S_{45})
Magnetic Field (B)


Entropy (S_{46})
Magnetic Field (B)


Entropy (S_{47})
Magnetic Field (B)


Entropy (S_{48})
Magnetic Field (B)


Entropy (S_{56})
Magnetic Field (B)


Entropy (S_{57})
Magnetic Field (B)

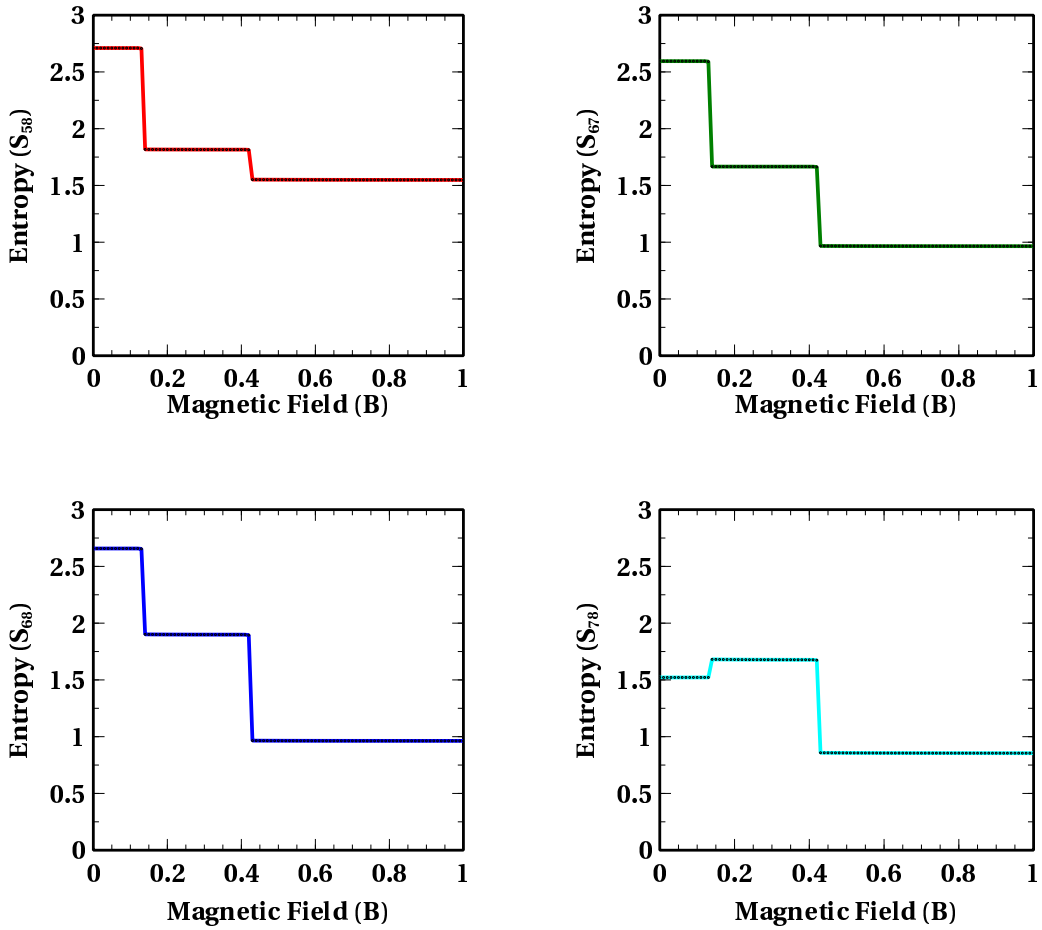
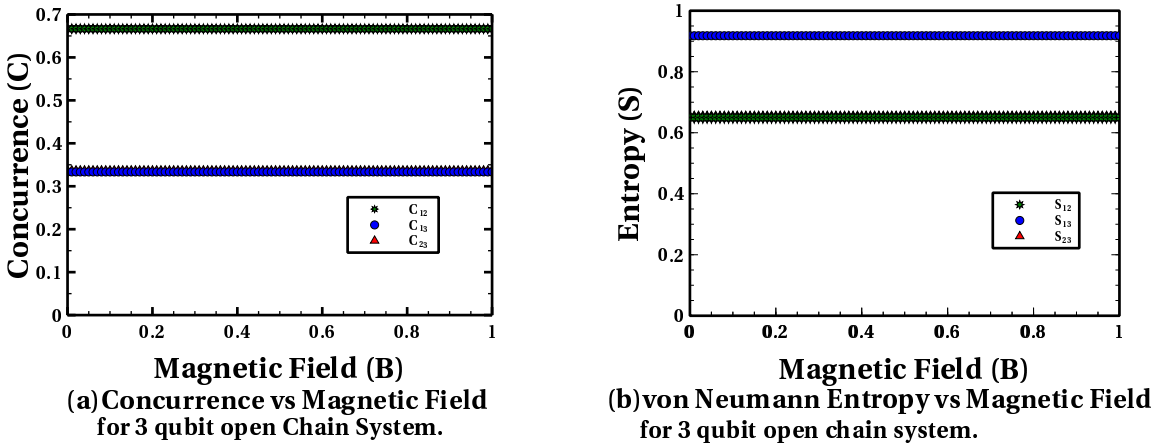


Figure 5.14: Variation of von Neumann Entropy with the Magnetic Field corresponding to 8-qubit periodic chain Heisenberg-like system.

5.3.3 Monogamy of Entanglement(MOE)

We can see that for both open chain as well as periodic chain configurations, qubits in the Heisenberg-Fibonacci spin chain systems follow the Monogamy of Entanglement(MOE) as evident from the Fig.[5.15] and Fig.[5.16]. We can see from the figures that the concurrences corresponding to the nearest neighbour (1,2) and (2,3) are very high, whereas that for (1,3) is not so. This implies the stronger entanglement of the (1,2) and (2,3) qubit pairs than the (1,3) qubit pairs. Again, the von Neumann entropy corresponding to the blocks containing (1,2) and (2,3) is small compared to the block containing (1,3) qubits, indicating the fact that the block containing (1,2) and (2,3) are less entangled with the other block containing rest number of qubits of the

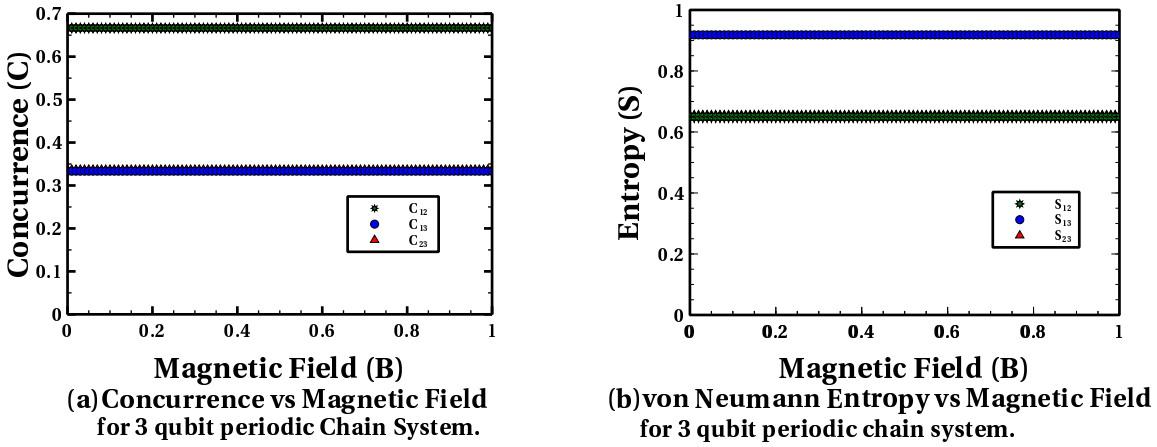
system. This thing is observed in both open as well as periodic chain configurations. We can also see that the 8-qubit system obeys MOE.



(a) Concurrence vs Magnetic Field for 3 qubit open Chain System.

(b) von Neumann Entropy vs Magnetic Field for 3 qubit open chain system.

Figure 5.15: Monogamy of Entanglement(MOE) for three-qubit open chain Heisenberg-Fibonacci system.



(a) Concurrence vs Magnetic Field for 3 qubit periodic Chain System.

(b) von Neumann Entropy vs Magnetic Field for 3 qubit periodic chain system.

Figure 5.16: Monogamy of Entanglement(MOE) for three-qubit periodic chain Heisenberg-Fibobsnni system.

5.4 Chapter Summary

In the previous chapter, we had discussed the Ising model involving the inverse Fibonacci type of interaction between the successive qubit pairs. After that, we were aiming to see what happens if we consider the same type of interaction scheme in the

Heisenberg model. With this motivation in mind, in the present chapter, we discussed the Heisenberg-Fibonacci system where the interaction strengths are varied like the inverse Fibonacci series. We have provided a smarter way with the bit test approach for constructing the Hamiltonian of such type of systems, which is a very efficient method. We also solved the Hamiltonian numerically and calculated their energy eigenvalue spectrum in both open as well as periodic chain configurations. We also discussed the entanglement study in this type of spin chain system by calculating the concurrences and von Neumann entropies for systems containing a smaller number of qubits (3 and 4) as well as systems containing a larger number of qubits (8-qubit system). We have observed some sharp discontinuities in concurrences and von Neumann entropies for the systems containing 4-qubit as well as higher numbers of qubits, and tried to explain the reason behind it. We have also observed that the monogamy of entanglement is observed in such a type of random Heisenberg-Fibonacci spin chain system, which is remarkable.

Chapter 6

Introduction to Bernoulli Numbers and Relation between Fibonacci and Bernoulli polynomials

---

6.1 Introduction

We are interested in studying how the relationship between two known polynomials and number series affects the entanglement between qubit pairs when we are studying a random spin chain system involving these number series for the first time. For that purpose, we had introduced the Fibonacci series in Chapter 2 and saw that by utilizing this number series, we can explain the entanglement between the qubits arranged like a quasicrystal structure. Now it is time to discuss another number series and utilize it in the random spin chain system, and analyze if we have some relation between the two numbers how the entanglement changes. We had chosen the Bernoulli series as the second number series to do our job because we have a relation between the Bernoulli series and the Fibonacci series, which is shown in this chapter.

6.2 What are Bernoulli numbers?

Bernoulli numbers [25][26] were introduced by Jakob Bernoulli. The main intention of mathematicians was to evaluate the sums of a large number of terms of large integer powers [27]. This means that they were very much eager to establish some general expression for this type of sum (sum of m^{th} powers of n positive integers) as can be shown below.

$$S_m(n) = 1^m + 2^m + 3^m + 4^m + ... + n^m \quad . \tag{6.1}$$

where n and m are positive integers. We know the sum for $m = 0, 1, 2, \ and \ 3$.

$$\begin{aligned}
S_1(n) &= 1^0 + 2^0 + 3^0 + 4^0 + ... + n^0 = n, \\
S_1(n) &= 1 + 2 + 3 + 4 + ... + n = \frac{n(n+1)}{2}, \\
S_2(n) &= 1^2 + 2^2 + 3^2 + 4^2 + ... + n^2 = \frac{n(n+1)(2n+1)}{6}, \\
S_3(n) &= 1^3 + 2^3 + 3^3 + 4^3 + ... + n^3 = \left(\frac{n(n+1)}{2}\right)^2,
\end{aligned}$$

What about $m > 3$? To answer this question, Bernoulli numbers were introduced. Let us arrange the summations containing $(n-1)$ number of terms in such a way that we can get some interesting pattern as follows,

$$\begin{aligned}
S_1(n-1) &= \frac{1}{2}n^2 - \frac{1}{2}n, \\
S_2(n-1) &= \frac{1}{3}n^3 - \frac{1}{2}n^2 + \frac{1}{6}n, \\
S_3(n-1) &= \frac{1}{4}n^4 - \frac{1}{2}n^3 + \frac{1}{4}n^2, \\
S_4(n-1) &= \frac{1}{5}n^5 - \frac{1}{2}n^4 + \frac{1}{3}n^3 - \frac{1}{30}, \\
S_5(n-1) &= \frac{1}{6}n^6 - \frac{1}{2}n^5 + \frac{5}{12}n^4 - \frac{1}{12}n^2, \\
S_6(n-1) &= \frac{1}{7}n^7 - \frac{1}{2}n^6 + \frac{1}{2}n^5 - \frac{1}{6}n^3 + \frac{1}{42}n, \\
&\vdots
\end{aligned}$$

Now we can get an idea about the leading term for the sum up to the m^{th} power like $S_m(n-1) = \frac{1}{m+1}n^{m+1}$. What about the remaining terms? For that, we can do the

simplification by taking out $\frac{1}{m+1}$ in each polynomial as a factor as below.

$$\begin{aligned}
S_1(n-1) &= \frac{1}{2}(n^2-n),\\
S_2(n-1) &= \frac{1}{3}(n^3-\frac{3}{2}n^2+\frac{1}{2}n),\\
S_3(n-1) &= \frac{1}{4}(n^4-2n^3+n^2),\\
S_4(n-1) &= \frac{1}{5}\left(n^5-\frac{5}{2}n^4+\frac{5}{3}n^3-\frac{1}{6}n\right),\\
&\vdots
\end{aligned}$$

Likewise we can express the other sum like $S_5(n-1)$, $S_6(n-1)$, $S_7(n-1)$ and so on. Now we can do further simplification as follows.

$$\begin{aligned}
S_1(n-1) &= \frac{1}{2}\left(n^2-2\left(\frac{1}{2}\right)n\right),\\
S_2(n-1) &= \frac{1}{3}\left(n^3-3\left(\frac{1}{2}\right)n^2+3\left(\frac{1}{6}\right)n\right),\\
S_3(n-1) &= \frac{1}{4}\left(n^4-4\left(\frac{1}{2}\right)n^3+6\left(\frac{1}{6}\right)n^2\right),\\
S_4(n-1) &= \frac{1}{5}\left(n^5-5\left(\frac{1}{2}\right)n^4+10\left(\frac{1}{6}\right)n^3-5\left(\frac{1}{30}\right)n\right),\\
&\vdots
\end{aligned}$$

Now we can find a pattern to express the sum. If I write one step, it will be clear to visualize the pattern.

$$\begin{aligned}
S_1(n-1) &= \frac{1}{2}\left({}^2C_0\ n^2-{}^2C_1\left(\frac{1}{2}\right)n\right),\\
S_2(n-1) &= \frac{1}{3}\left({}^3C_0\ n^3-{}^3C_1\left(\frac{1}{2}\right)n^2+{}^3C_2\left(\frac{1}{6}\right)n\right),\\
S_3(n-1) &= \frac{1}{4}\left({}^4C_0\ n^4-{}^4C_1\left(\frac{1}{2}\right)n^3+{}^4C_2\left(\frac{1}{6}\right)n^2+{}^4C_3(0)\ n\right),\\
S_4(n-1) &= \frac{1}{5}\left({}^5C_0\ n^5-{}^5C_1\left(\frac{1}{2}\right)n^4+{}^5C_2\left(\frac{1}{6}\right)n^3-{}^5C_3(0)n^2-{}^5C_4\left(\frac{1}{30}\right)n\right),\\
&\vdots
\end{aligned}$$

So we finally obtain a pattern as follows,

$$S_m(n-1) = \frac{1}{m+1}\sum_{k=0}^{m} {}^{m+1}C_k\ B_k\ n^{m-k+1}. \tag{6.2}$$

Here, the B_k's are the Bernoulli numbers, which are given as,

$$B_0 = 1, \quad B_1 = -\frac{1}{2}, \quad B_2 = \frac{1}{6}, \quad B_3 = 0, \quad B_4 = -\frac{1}{30}, \quad B_5 = 0, \quad B_6 = \frac{1}{42} \text{ and so on.}$$

These numbers can be generated easily by the generating function $G(x) = \dfrac{x}{e^x - 1}$ as proposed by Euler, which can be given as follows,

$$\boxed{\sum_{k=0}^{\infty} B_k \frac{x^k}{k!} = \frac{x}{e^x - 1}}. \tag{6.3}$$

The Bernoulli numbers have the speciality that the odd Bernoulli numbers are 0 (i.e., $B_{2k+1} = 0$, excluding $B_1 = -\frac{1}{2}$). We can observe this from the sum expressions mentioned above, and starting from the generating function, we can also prove this property as follows.

$$\frac{x}{e^x - 1} = B_0 + B_1\, x + B_2 \frac{x^2}{2!} + B_3 \frac{x^3}{3!} + ...$$

Now we can write,

$$\begin{aligned} g(x) &= \frac{x}{e^x - 1} - B_1\, x, \\ &= \frac{2x + x(e^x - 1)}{2(e^x - 1)}, \\ &= \frac{x + xe^x}{2(e^x - 1)}. \end{aligned}$$

$$\begin{aligned} g(x) &= \frac{x(1 + e^x)}{2(e^x - 1)}, \\ &= \frac{x}{2} \frac{(e^x + 1)}{(e^x - 1)}, \\ &= \frac{x}{2} \frac{(e^x + 1)}{(e^x - 1)} \frac{e^{\frac{-x}{2}}}{e^{\frac{-x}{2}}}, \\ &= \frac{x}{2} \left(\frac{e^{\frac{x}{2}} + e^{\frac{-x}{2}}}{e^{\frac{x}{2}} - e^{\frac{-x}{2}}} \right). \end{aligned}$$

Now,

$$\begin{aligned} g(-x) &= \frac{-x}{2} \left(\frac{e^{\frac{-x}{2}} + e^{\frac{x}{2}}}{e^{\frac{-x}{2}} - e^{\frac{x}{2}}} \right), \\ &= \frac{x}{2} \left(\frac{e^{\frac{x}{2}} + e^{\frac{x}{2}}}{e^{\frac{x}{2}} - e^{\frac{-x}{2}}} \right), \\ &= g(x). \end{aligned} \tag{6.4}$$

Therefore, only even terms will exist from the series given by Wq.[6.3] and the odd terms are zero, excluding B_1, which is $-\frac{1}{2}$.

6.3 Determination of Bernoulli numbers from the generating function

We already saw that from the generating function we can write,

$$f(x) = \frac{x}{e^x - 1} = B_0 + B_1\, x + B_2 \frac{x^2}{2!} + B_3 \frac{x^3}{3!} + ...$$

Therefore, we can find B_0, B_1, B_3,... etc. as follows.

$$\begin{aligned}
B_0 &= \lim_{x\to 0} f(x), \\
&= \lim_{x\to 0} \frac{x}{e^x - 1}, && \left[\frac{0}{0} \text{ form, we have to use L'Hospitial rule}\right] \\
&= \lim_{x\to 0} \frac{1}{e^x}, && \text{[using L'Hospitial rule]} \\
&= 1.
\end{aligned}$$

$$\begin{aligned}
B_1 &= \lim_{x\to 0} f'(x), \\
&= \lim_{x\to 0} \left[\frac{1}{(e^x - 1)} - \frac{xe^x}{(e^x - 1)^2}\right], \\
&= \lim_{x\to 0} \frac{e^x - 1 - xe^x}{(e^x - 1)^2}, && \text{[Again} \frac{0}{0} \text{ form]} \\
&= \lim_{x\to 0} \frac{e^x - e^x - xe^x}{2(e^x - 1)e^x}, && \text{[using L'Hospitial rule]} \\
&= \lim_{x\to 0} \frac{e^x - e^x - e^x - xe^x}{2(e^x - 1)e^x + 2e^{2x}}, && \text{[Again using L'Hospitial rule]} \\
&= -\frac{1}{2}.
\end{aligned}$$

$$
\begin{aligned}
B_2 &= \lim_{x\to 0} f''(x), \\
&= \lim_{x\to 0}\left[\frac{e^x - e^x - xe^x}{(e^x-1)^2} - \frac{2(e^x - 1 - xe^x)e^x}{(e^x-1)^3}\right], \\
&= \lim_{x\to 0}\left[\frac{xe^{2x} + xe^x - 2e^{2x} + 2e^x}{(e^x-1)^3}\right], && [\text{Again}\frac{0}{0}\ \text{form}] \\
&= \lim_{x\to 0}\left[\frac{2xe^{2x} + e^{2x} + 2e^x - 4e^{2x} + e^x + xe^x}{3(e^x-1)^2\ e^x}\right], && [\text{using L'Hospitial rule}] \\
&= \lim_{x\to 0}\left[\frac{2e^{2x} + 4xe^{2x} + 2e^{2x} + 2e^x - 8e^{2x} + e^x + e^x + xe^x}{3e^x(e^x-1)^2 + 6e^x(e^x-1)e^x}\right], && [\text{using L'Hospitial rule}] \\
&= \lim_{x\to 0}\left[\frac{4e^{2x} + 4e^{2x} + 8xe^{2x} - 12e^{2x} + 4e^x + e^x + xe^x}{3e^x(e^x-1)^2 + 6e^x(e^x-1)e^x + 12e^{2x}(e^x-1) + 6e^{3x}}\right], && [\text{using L'Hospitial rule}] \\
&= \left[\frac{4+4-12+4+1}{6}\right], \\
&= \frac{1}{6}.
\end{aligned}
$$

We will not evaluate further Bernoulli numbers by following the same procedure by hand, but we can use Mathematica. For example, we can determine B_3 by taking the limiting value of the 3rd-order derivative of $f(x)$. Let us take, $u(x) = f''(x) = \frac{xe^{2x} + xe^x - 2e^{2x} + 2e^x}{(e^x-1)^3}$. Now we do further calculations through Mathematica as follows.

Bernoulli's Number evaluation

3rd Bernoulli number

In[1]:=
```
u[x_] = (x * Exp[2 * x] + 2 * Exp[x] - 2 * Exp[2 * x] + x * Exp[x]) / (Exp[x] - 1) ^ 3
u '[x]
```

Out[1]=
$$\frac{2 e^x - 2 e^{2x} + e^x x + e^{2x} x}{(-1 + e^x)^3}$$

Out[2]=
$$-\frac{3 e^x \left(2 e^x - 2 e^{2x} + e^x x + e^{2x} x\right)}{(-1 + e^x)^4} + \frac{3 e^x - 3 e^{2x} + e^x x + 2 e^{2x} x}{(-1 + e^x)^3}$$

In[3]:=
```
Simplify[-(3 e^x (2 e^x - 2 e^(2 x) + e^x x + e^(2 x) x))/(-1 + e^x)^4 + (3 e^x - 3 e^(2 x) + e^x x + 2 e^(2 x) x)/(-1 + e^x)^3]
```

Out[3]=
$$-\frac{e^x \left(3 + e^{2x} (-3 + x) + x + 4 e^x x\right)}{(-1 + e^x)^4}$$

Numerator

In[4]:=
```
u1[x_] = - e^x (3 + e^(2 x) (-3 + x) + x + 4 e^x x)
```

Out[4]=
$$-e^x \left(3 + e^{2x} (-3 + x) + x + 4 e^x x\right)$$

In[5]:=
```
∂x (-e^x (3 + e^(2 x) (-3 + x) + x + 4 e^x x))
```

Out[5]=
$$-e^x \left(1 + 4 e^x + e^{2x} + 2 e^{2x} (-3 + x) + 4 e^x x\right) - e^x \left(3 + e^{2x} (-3 + x) + x + 4 e^x x\right)$$

In[6]:=
```
∂x (-e^x (1 + 4 e^x + e^(2 x) + 2 e^(2 x) (-3 + x) + 4 e^x x) - e^x (3 + e^(2 x) (-3 + x) + x + 4 e^x x))
```

Out[6]=
$$-2 e^x \left(1 + 4 e^x + e^{2x} + 2 e^{2x} (-3 + x) + 4 e^x x\right) - e^x \left(8 e^x + 4 e^{2x} + 4 e^{2x} (-3 + x) + 4 e^x x\right) - e^x \left(3 + e^{2x} (-3 + x) + x + 4 e^x x\right)$$

In[7]:=
$$\partial_x\left(-2e^x\left(1+4e^x+e^{2x}+2e^{2x}(-3+x)+4e^x x\right)-e^x\left(8e^x+4e^{2x}+4e^{2x}(-3+x)+4e^x x\right)-e^x\left(3+e^{2x}(-3+x)+x+4e^x x\right)\right)$$

Out[7]=
$$-3e^x\left(1+4e^x+e^{2x}+2e^{2x}(-3+x)+4e^x x\right)-3e^x\left(8e^x+4e^{2x}+4e^{2x}(-3+x)+4e^x x\right)-e^x\left(12e^x+12e^{2x}+8e^{2x}(-3+x)+4e^x x\right)-e^x\left(3+e^{2x}(-3+x)+x+4e^x x\right)$$

In[8]:=
$$\partial_x\left(-3e^x\left(1+4e^x+e^{2x}+2e^{2x}(-3+x)+4e^x x\right)-3e^x\left(8e^x+4e^{2x}+4e^{2x}(-3+x)+4e^x x\right)-e^x\left(12e^x+12e^{2x}+8e^{2x}(-3+x)+4e^x x\right)-e^x\left(3+e^{2x}(-3+x)+x+4e^x x\right)\right)$$

Out[8]=
$$-4e^x\left(1+4e^x+e^{2x}+2e^{2x}(-3+x)+4e^x x\right)-6e^x\left(8e^x+4e^{2x}+4e^{2x}(-3+x)+4e^x x\right)-4e^x\left(12e^x+12e^{2x}+8e^{2x}(-3+x)+4e^x x\right)-e^x\left(16e^x+32e^{2x}+16e^{2x}(-3+x)+4e^x x\right)-e^x\left(3+e^{2x}(-3+x)+x+4e^x x\right)$$

In[9]:=
$$\text{Limit}\left[-4e^x\left(1+4e^x+e^{2x}+2e^{2x}(-3+x)+4e^x x\right)-6e^x\left(8e^x+4e^{2x}+4e^{2x}(-3+x)+4e^x x\right)-4e^x\left(12e^x+12e^{2x}+8e^{2x}(-3+x)+4e^x x\right)-e^x\left(16e^x+32e^{2x}+16e^{2x}(-3+x)+4e^x x\right)-e^x\left(3+e^{2x}(-3+x)+x+4e^x x\right),\ x\to 0\right]$$

Out[9]=
$$0$$

Denominator

In[10]:=
$$h1[x_] = (-1+e^x)^4$$

Out[10]=
$$(-1+e^x)^4$$

In[11]:=
$$\partial_x(-1+e^x)^4$$

Out[11]=
$$4e^x(-1+e^x)^3$$

In[12]:=
$$\partial_x\left(4e^x(-1+e^x)^3\right)$$

Out[12]=
$$12e^{2x}(-1+e^x)^2+4e^x(-1+e^x)^3$$

In[13]:=
$$\partial_x\left(12e^{2x}(-1+e^x)^2+4e^x(-1+e^x)^3\right)$$

Out[13]=
$$24e^{3x}(-1+e^x)+36e^{2x}(-1+e^x)^2+4e^x(-1+e^x)^3$$

In[14]:=
$$\partial_x\left(24e^{3x}(-1+e^x)+36e^{2x}(-1+e^x)^2+4e^x(-1+e^x)^3\right)$$

Out[14]=
$$24e^{4x}+144e^{3x}(-1+e^x)+84e^{2x}(-1+e^x)^2+4e^x(-1+e^x)^3$$

In[15]:=
```
Limit[24 e^(4 x) + 144 e^(3 x) (-1 + e^x) + 84 e^(2 x) (-1 + e^x)^2 + 4 e^x (-1 + e^x)^3, x → 0]
```

Out[15]=
```
24
```

3rd Bernoulli number

In[17]:=
```
B3 = Limit[-4 e^x (1 + 4 e^x + e^(2 x) + 2 e^(2 x) (-3 + x) + 4 e^x x) -
    6 e^x (8 e^x + 4 e^(2 x) + 4 e^(2 x) (-3 + x) + 4 e^x x) - 4 e^x (12 e^x + 12 e^(2 x) + 8 e^(2 x) (-3 + x) + 4 e^x x) -
    e^x (16 e^x + 32 e^(2 x) + 16 e^(2 x) (-3 + x) + 4 e^x x) - e^x (3 + e^(2 x) (-3 + x) + x + 4 e^x x), x → 0] /
  Limit[24 e^(4 x) + 144 e^(3 x) (-1 + e^x) + 84 e^(2 x) (-1 + e^x)^2 + 4 e^x (-1 + e^x)^3, x → 0]
```

Out[17]=
```
0
```

4th Bernoulli number

In[18]:=
```
t[x_] = -(e^x (3 + e^(2 x) (-3 + x) + x + 4 e^x x)) / (-1 + e^x)^4

t'[x]
```

Out[18]=
$$-\frac{e^x \left(3+e^{2 x} (-3+x)+x+4 e^x x\right)}{\left(-1+e^x\right)^4}$$

Out[19]=
$$-\frac{e^x \left(1+4 e^x+e^{2 x}+2 e^{2 x} (-3+x)+4 e^x x\right)}{\left(-1+e^x\right)^4}+\frac{4 e^{2 x} \left(3+e^{2 x} (-3+x)+x+4 e^x x\right)}{\left(-1+e^x\right)^5}-\frac{e^x \left(3+e^{2 x} (-3+x)+x+4 e^x x\right)}{\left(-1+e^x\right)^4}$$

In[20]:=
```
Simplify[-(e^x (1 + 4 e^x + e^(2 x) + 2 e^(2 x) (-3 + x) + 4 e^x x)) / (-1 + e^x)^4 +
  (4 e^(2 x) (3 + e^(2 x) (-3 + x) + x + 4 e^x x)) / (-1 + e^x)^5 - (e^x (3 + e^(2 x) (-3 + x) + x + 4 e^x x)) / (-1 + e^x)^4]
```

Out[20]=
$$\frac{e^x \left(4+e^{3 x} (-4+x)+x+e^{2 x} (-12+11 x)+e^x (12+11 x)\right)}{\left(-1+e^x\right)^5}$$

Numerator

In[21]:=
$$t1[x_] = e^x\left(4 + e^{3x}(-4+x) + x + e^{2x}(-12+11x) + e^x(12+11x)\right)$$

Out[21]=
$$e^x\left(4 + e^{3x}(-4+x) + x + e^{2x}(-12+11x) + e^x(12+11x)\right)$$

In[22]:=
$$\partial_x\left(e^x\left(4 + e^{3x}(-4+x) + x + e^{2x}(-12+11x) + e^x(12+11x)\right)\right)$$

Out[22]=
$$e^x\left(4 + e^{3x}(-4+x) + x + e^{2x}(-12+11x) + e^x(12+11x)\right) + e^x\left(1 + 11e^x + 11e^{2x} + e^{3x} + 3e^{3x}(-4+x) + 2e^{2x}(-12+11x) + e^x(12+11x)\right)$$

In[23]:=
$$\partial_x\left(e^x\left(4 + e^{3x}(-4+x) + x + e^{2x}(-12+11x) + e^x(12+11x)\right) + e^x\left(1 + 11e^x + 11e^{2x} + e^{3x} + 3e^{3x}(-4+x) + 2e^{2x}(-12+11x) + e^x(12+11x)\right)\right)$$

Out[23]=
$$e^x\left(4 + e^{3x}(-4+x) + x + e^{2x}(-12+11x) + e^x(12+11x)\right) + 2e^x\left(1 + 11e^x + 11e^{2x} + e^{3x} + 3e^{3x}(-4+x) + 2e^{2x}(-12+11x) + e^x(12+11x)\right) + e^x\left(22e^x + 44e^{2x} + 6e^{3x} + 9e^{3x}(-4+x) + 4e^{2x}(-12+11x) + e^x(12+11x)\right)$$

In[24]:=
$$\partial_x \%23$$

Out[24]=
$$e^x\left(4 + e^{3x}(-4+x) + x + e^{2x}(-12+11x) + e^x(12+11x)\right) + 3e^x\left(1 + 11e^x + 11e^{2x} + e^{3x} + 3e^{3x}(-4+x) + 2e^{2x}(-12+11x) + e^x(12+11x)\right) + 3e^x\left(22e^x + 44e^{2x} + 6e^{3x} + 9e^{3x}(-4+x) + 4e^{2x}(-12+11x) + e^x(12+11x)\right) + e^x\left(33e^x + 132e^{2x} + 27e^{3x} + 27e^{3x}(-4+x) + 8e^{2x}(-12+11x) + e^x(12+11x)\right)$$

In[25]:=
$$\partial_x \%24$$

Out[25]=
$$e^x\left(4 + e^{3x}(-4+x) + x + e^{2x}(-12+11x) + e^x(12+11x)\right) + 4e^x\left(1 + 11e^x + 11e^{2x} + e^{3x} + 3e^{3x}(-4+x) + 2e^{2x}(-12+11x) + e^x(12+11x)\right) + 6e^x\left(22e^x + 44e^{2x} + 6e^{3x} + 9e^{3x}(-4+x) + 4e^{2x}(-12+11x) + e^x(12+11x)\right) + 4e^x\left(33e^x + 132e^{2x} + 27e^{3x} + 27e^{3x}(-4+x) + 8e^{2x}(-12+11x) + e^x(12+11x)\right) + e^x\left(44e^x + 352e^{2x} + 108e^{3x} + 81e^{3x}(-4+x) + 16e^{2x}(-12+11x) + e^x(12+11x)\right)$$

In[26]:= $\partial_x \%25$

Out[26]=

$$e^x \left(4 + e^{3x} (-4 + x) + x + e^{2x} (-12 + 11x) + e^x (12 + 11x)\right) +$$
$$5\, e^x \left(1 + 11\, e^x + 11\, e^{2x} + e^{3x} + 3\, e^{3x} (-4 + x) + 2\, e^{2x} (-12 + 11x) + e^x (12 + 11x)\right) +$$
$$10\, e^x \left(22\, e^x + 44\, e^{2x} + 6\, e^{3x} + 9\, e^{3x} (-4 + x) + 4\, e^{2x} (-12 + 11x) + e^x (12 + 11x)\right) +$$
$$10\, e^x \left(33\, e^x + 132\, e^{2x} + 27\, e^{3x} + 27\, e^{3x} (-4 + x) + 8\, e^{2x} (-12 + 11x) + e^x (12 + 11x)\right) +$$
$$5\, e^x \left(44\, e^x + 352\, e^{2x} + 108\, e^{3x} + 81\, e^{3x} (-4 + x) + 16\, e^{2x} (-12 + 11x) + e^x (12 + 11x)\right) +$$
$$e^x \left(55\, e^x + 880\, e^{2x} + 405\, e^{3x} + 243\, e^{3x} (-4 + x) + 32\, e^{2x} (-12 + 11x) + e^x (12 + 11x)\right)$$

In[27]:= $\texttt{Limit}\left[e^x \left(4 + e^{3x} (-4 + x) + x + e^{2x} (-12 + 11x) + e^x (12 + 11x)\right) + \right.$
$5\, e^x \left(1 + 11\, e^x + 11\, e^{2x} + e^{3x} + 3\, e^{3x} (-4 + x) + 2\, e^{2x} (-12 + 11x) + e^x (12 + 11x)\right) +$
$10\, e^x \left(22\, e^x + 44\, e^{2x} + 6\, e^{3x} + 9\, e^{3x} (-4 + x) + 4\, e^{2x} (-12 + 11x) + e^x (12 + 11x)\right) +$
$10\, e^x \left(33\, e^x + 132\, e^{2x} + 27\, e^{3x} + 27\, e^{3x} (-4 + x) + 8\, e^{2x} (-12 + 11x) + e^x (12 + 11x)\right) +$
$5\, e^x \left(44\, e^x + 352\, e^{2x} + 108\, e^{3x} + 81\, e^{3x} (-4 + x) + 16\, e^{2x} (-12 + 11x) + e^x (12 + 11x)\right) +$
$\left. e^x \left(55\, e^x + 880\, e^{2x} + 405\, e^{3x} + 243\, e^{3x} (-4 + x) + 32\, e^{2x} (-12 + 11x) + e^x (12 + 11x)\right), x \to 0\right]$

Out[27]=

-4

Denominator

In[28]:= $\texttt{t2[x_] = } (-1 + e^x)^5$

Out[28]=

$(-1 + e^x)^5$

In[29]:= $\partial_x (-1 + e^x)^5$

Out[29]=

$5\, e^x (-1 + e^x)^4$

In[30]:= $\partial_x \left(5\, e^x (-1 + e^x)^4\right)$

Out[30]=

$20\, e^{2x} (-1 + e^x)^3 + 5\, e^x (-1 + e^x)^4$

In[31]:= $\partial_x \left(20\, e^{2x} (-1 + e^x)^3 + 5\, e^x (-1 + e^x)^4\right)$

Out[31]=

$60\, e^{3x} (-1 + e^x)^2 + 60\, e^{2x} (-1 + e^x)^3 + 5\, e^x (-1 + e^x)^4$

In[32]:=
$$\partial_x\left(60\,e^{3x}(-1+e^x)^2+60\,e^{2x}(-1+e^x)^3+5\,e^x(-1+e^x)^4\right)$$

Out[32]=
$$120\,e^{4x}(-1+e^x)+360\,e^{3x}(-1+e^x)^2+140\,e^{2x}(-1+e^x)^3+5\,e^x(-1+e^x)^4$$

In[33]:=
$$\partial_x\left(120\,e^{4x}(-1+e^x)+360\,e^{3x}(-1+e^x)^2+140\,e^{2x}(-1+e^x)^3+5\,e^x(-1+e^x)^4\right)$$

Out[33]=
$$120\,e^{5x}+1200\,e^{4x}(-1+e^x)+1500\,e^{3x}(-1+e^x)^2+300\,e^{2x}(-1+e^x)^3+5\,e^x(-1+e^x)^4$$

In[34]:=
```
Limit[120 e^(5 x) + 1200 e^(4 x) (-1 + e^x) + 1500 e^(3 x) (-1 + e^x)^2 + 300 e^(2 x) (-1 + e^x)^3 + 5 e^x (-1 + e^x)^4, x → 0]
```

Out[34]=
$$120$$

4th Bernoulli number

In[35]:=
```
B4 = Limit[e^x (4 + e^(3 x) (-4 + x) + x + e^(2 x) (-12 + 11 x)) + e^x (12 + 11 x)) +
      5 e^x (1 + 11 e^x + 11 e^(2 x) + e^(3 x) + 3 e^(3 x) (-4 + x) + 2 e^(2 x) (-12 + 11 x)) + e^x (12 + 11 x)) +
      10 e^x (22 e^x + 44 e^(2 x) + 6 e^(3 x) + 9 e^(3 x) (-4 + x) + 4 e^(2 x) (-12 + 11 x)) + e^x (12 + 11 x)) +
      10 e^x (33 e^x + 132 e^(2 x) + 27 e^(3 x) + 27 e^(3 x) (-4 + x) + 8 e^(2 x) (-12 + 11 x)) + e^x (12 + 11 x)) +
      5 e^x (44 e^x + 352 e^(2 x) + 108 e^(3 x) + 81 e^(3 x) (-4 + x) + 16 e^(2 x) (-12 + 11 x)) + e^x (12 + 11 x)) +
      e^x (55 e^x + 880 e^(2 x) + 405 e^(3 x) + 243 e^(3 x) (-4 + x) + 32 e^(2 x) (-12 + 11 x)) + e^x (12 + 11 x)), x → 0] /
    Limit[120 e^(5 x) + 1200 e^(4 x) (-1 + e^x) + 1500 e^(3 x) (-1 + e^x)^2 + 300 e^(2 x) (-1 + e^x)^3 + 5 e^x (-1 + e^x)^4, x → 0]
```

Out[35]=
$$-\frac{1}{30}$$

6.4 Recurrence relation for Bernoulli number

We have observed that it is very laborious to extract the Bernoulli numbers from the generating function. So we can use the recurrence relation to generate the Bernoulli numbers, which is a very effective method. We can start from the generating function from Eq.[6.3] as follows,

$$\begin{aligned}
\frac{x}{e^x-1} &= \sum_{i=0}^{\infty} B_i \frac{x^i}{i!}, \\
x &= (e^x-1)\sum_{i=0}^{\infty} B_i \frac{x^i}{i!}, \\
&= \left(1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots - 1\right)\sum_{i=0}^{\infty} B_i \frac{x^i}{i!}, \\
&= \sum_{j=1}^{\infty} \frac{x^j}{j!} \sum_{i=0}^{\infty} B_i \frac{x^i}{i!}.
\end{aligned}$$

$$\frac{x}{e^x-1} = \sum_{j=0}^{\infty} \frac{x^{j+1}}{(j+1)!} \sum_{i=0}^{\infty} B_i \frac{x^i}{i!},$$

[Using Cauchy product of infinite sum:

$$\left(\sum_{k=0}^{\infty} a_k\right)\left(\sum_{m=0}^{\infty} b_m\right) = \sum_{n=0}^{\infty} c_n,$$

where $c_n = a_0 b_n + a_1 b_{n-1} + \cdots + a_n b_0 = \sum_{k=0}^{n} a_k b_{n-k}$]

$$\begin{aligned}
&= \sum_{n=0}^{\infty}\sum_{k=0}^{n} \frac{x^{n+1-k}}{(n+1-k)!}\frac{B_k\ x^k}{k!}, \\
&= \sum_{n=0}^{\infty}\sum_{k=0}^{n} \frac{(n+1)!}{(n+1-k)!k!}\ B_k \frac{x^{n+1}}{(n+1)!}, \\
&= \sum_{n=0}^{\infty}\sum_{k=0}^{n} {}^{n+1}C_k\ B_k \frac{x^{n+1}}{(n+1)!}, \\
x &= \sum_{n=1}^{\infty}\sum_{k=0}^{n-1} {}^{n}C_k\ B_k \frac{x^n}{n!}. \qquad (6.5)
\end{aligned}$$

Now, from this Eq.[6.5] for $n = 1$, $B_0 = 1$ and for $n > 1$ we are getting

$$\boxed{\sum_{k=0}^{n-1} {}^{n}C_k\ B_k = 0} \qquad (6.6)$$

Expanding Eq.[6.6] we can get as follows,

$$\begin{aligned} B_0 &= 1 && \text{(for } n = 1\text{)}, \\ B_0 + 2B_1 &= 0 && \text{(for } n = 2\text{)}, \\ B_0 + 3B_1 + 3B_2 &= 0 && \text{(for } n = 3\text{)}, \\ &\vdots \end{aligned}$$

From these equations, we can write the following,

$$\begin{aligned} B_1 &= -\frac{1}{2}B_0 = -\frac{1}{2}{}^2C_0\ B_0, \\ B_2 &= -\frac{1}{3}(B_0 + 3B_1) = -\frac{1}{3}({}^3C_0 B_0 + {}^3C_1\ B_1), \\ B_3 &= -\frac{1}{4}(B_0 + 4B_1 + 6B_2) = -\frac{1}{4}({}^4C_0 B_0 + {}^4C_1 B_1 + {}^4C_2 B_2), \\ &\vdots \end{aligned}$$

Therefore, in general, we can write,

$$B_n = -\frac{1}{n+1}\sum_{k=0}^{n-1} {}^{n+1}C_k\ B_k \quad . \tag{6.7}$$

6.5 Numerical generation of Bernoulli Numbers

We can generate the Bernoulli numbers numerically using the Eq.[6.7]. Here we have with the Fortran code to generate the Bernoulli numbers.

```
program Bernoulli_number
    implicit none
    double precision, allocatable :: B(:)
    integer :: n, i, j, k
    double precision :: sum

    ! Ask the user to give the number of terms
    print*, "Enter the number of terms for Bernoulli series:"
    read*, n
    if (n <= 0) then
        print*, "Number of terms must be greater than 0."
```

```
        stop
    end if

    ! Allocate the array
    allocate(B(0:n))

    ! Initialization of the first term for the series
    B(0) = 1.0d0  ! B(0) = 1

    ! Calculate the Bernoulli numbers
    do j = 1, n
        sum = 0.0d0
        do k = 0, j - 1
            sum = sum + (binomial_coefficient(j + 1, k)  * B(k))
        end do
        B(j) = -sum / (j + 1.0d0)
    end do

    ! Print the Bernoulli series
    print*, "Bernoulli series:"
    do i = 0, n
        write(91,*) i, B(i)
    end do

    deallocate(B)

contains

    ! Function to calculate binomial coefficient
    function binomial_coefficient(n, k) result(coefficient)
        implicit none
        integer, intent(in) :: n, k
        double precision :: coefficient
        integer :: i
        coefficient = 1.0d0
        if (k > 0 .and. k < n) then
            do i = 1, k
                coefficient = coefficient * (n - i + 1) / i
```

```
            end do
        else if (k == 0 .or. k == n) then
            coefficient = 1.0d0
        else
            coefficient = 0.0d0
        end if
    end function binomial_coefficient

end program Bernoulli_number
```

We have listed the first 14 of the Bernoulli numbers obtained by implementing the code and plotted them with the number of terms, which is shown below in Fig. [6.1].

6.5.0.1 List of first 14 Bernoulli numbers

Here, the first column indicates the term number, and the second column shows the respective Bernoulli number up to double precision.

```
            0    1.0000000000000000
            1 -0.50000000000000000
            2  0.16666666666666666
            3  -0.0000000000000000
            4  -3.3333333333333305E-002
            5  -7.4014868308343765E-017
            6   2.3809523809523885E-002
            7   6.9388939039072284E-017
            8  -3.3333333333333638E-002
            9   1.7763568394002506E-016
           10   7.5757575757576454E-002
           11  -1.4802973661668755E-015
           12 -0.25311355311355305
           13   4.3140094671148936E-015
           14   1.1666666666666603
```

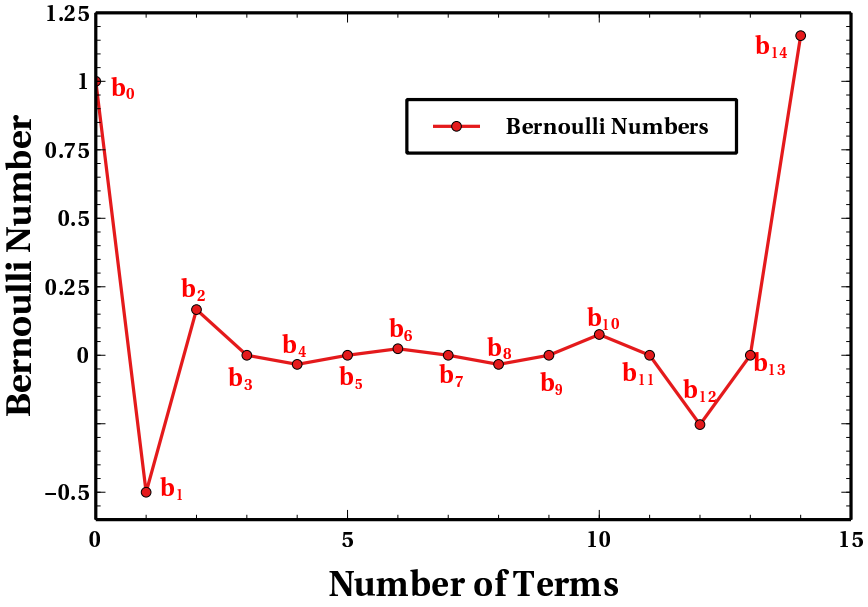


Figure 6.1: Plotting of the first 14 Bernoulli numbers with the number of terms.

6.6 Relations and Identities of Bernoulli Polynomials

We want to relate the Bernoulli series as well as the Bernoulli polynomials with the Fibonacci series (discussed in chapter 2), as well as Fibonacci polynomials (will be discussed in the next section), which can lead us to the study of how the entanglement between qubits are affected when they are arranged in a random spin chain system involving these type of number series as interaction strengths.

6.6.1 Generating Function Of Bernoulli Number and Bernoulli Polynomial

The generating function of Bernoulli Numbers($\{ B_k \}$) can be given by,

$$\sum_{k=0}^{\infty} B_k \frac{x^k}{k!} = \frac{x}{e^x - 1}. \tag{6.8}$$

The generating function of Bernoulli polynomials can be given by,

$$\sum_{n=0}^{\infty} B_n(t)\frac{x^n}{n!} = \frac{xe^{tx}}{e^x - 1}. \tag{6.9}$$

6.6.2 Relation Between Bernoulli Polynomial and Bernoulli Number

We can start from the generating function of the Bernoulli polynomial that can be given by,

$$\sum_{n=0}^{\infty} B_n(t)\frac{x^n}{n!} = \frac{xe^{tx}}{e^x - 1}. \tag{6.10}$$

We can now modify the R.H.S of this equation as follows,

$$\begin{aligned}\sum_{n=0}^{\infty} B_n(t)\frac{x^n}{n!} &= \frac{xe^{tx}}{e^x - 1},\\ &= (\frac{x}{e^x - 1})e^{tx},\\ &= \sum_{i=0}^{\infty} B_i\frac{x^i}{i!}\sum_{j=0}^{\infty}\frac{(tx)^j}{j!}.\end{aligned} \tag{6.11}$$

Let us use Cauchy's product of two infinite series, which can be given as follows,

$$(\sum_{k=0}^{\infty} a_k)(\sum_{m=0}^{\infty} b_m) = (\sum_{n=0}^{\infty} c_n). \tag{6.12}$$

where,

$$\begin{aligned}c_n &= a_0b_n + a_1b_{n-1} + ... + a_nb_0,\\ &= \sum_{k=0}^{n} a_kb_{n-k}.\end{aligned} \tag{6.13}$$

From Eq.[6.11], using Eq.[6.12],

$$\begin{aligned}\sum_{n=0}^{\infty} B_n(t)\frac{x^n}{n!} &= \sum_{n=0}^{\infty}\sum_{k=0}^{n}\frac{(tx)^{n-k}}{(n-k)!}B_k\frac{x^k}{k!},\\ &= \sum_{n=0}^{\infty}\sum_{k=0}^{n}\frac{n!}{(n-k)!k!}t^{n-k}B_k\frac{x^n}{n!}.\end{aligned} \tag{6.14}$$

Therefore comparing the coefficient of $\frac{x^n}{n!}$ on both sides, we can get

$$B_n(t) = \sum_{k=0}^{n} {}^nC_k\ B_kt^{n-k}. \tag{6.15}$$

6.6.3 An interesting Relation

Using the general recursion relation as mentioned in Eq.[6.9] we can write,

$$\begin{aligned} B_n(x+1) - B_n(x) &= \frac{ze^{(x+1)z}}{e^z - 1} - \frac{ze^{xz}}{e^z - 1}, \\ &= \frac{ze^{xz}(e^z - 1)}{(e^z - 1)}, \\ &= ze^{xz}. \end{aligned} \tag{6.16}$$

$$\sum_{n=0}^{\infty} (B_n(x+1) - B_n(x)) = \sum_{n=0}^{\infty} \frac{x^n}{n!} z^{n+1}. \tag{6.17}$$

Now, equating the coefficient of z^n from both sides,

$$B_n(x+1) - B_n(x) = nx^{n-1}. \tag{6.18}$$

6.6.4 Recursion Relation of the Bernoulli polynomial

Let us discuss the recurrence relation of the Bernoulli polynomial. From Eq.[6.9]

$$\sum_{n=0}^{\infty} B_n(x) \frac{z^n}{n!} = \frac{ze^{zx}}{e^z - 1}. \tag{6.19}$$

Differentiating both sides with respect to x we get,

$$\begin{aligned} \sum_{n=0}^{\infty} B_n'(x) \frac{z^n}{n!} &= z \frac{ze^{zx}}{e^z - 1}, \\ &= \sum_{n=0}^{\infty} B_n(x) \frac{z^{n+1}}{n!}. \end{aligned} \tag{6.20}$$

Equating the coefficient of z^n from both sides we get,

$$B_n'(x) = nB_{n-1}(x). \tag{6.21}$$

Putting $n = n + 1$,

$$B_{n+1}'(x) = (n+1)B_n(x). \tag{6.22}$$

6.6.5 Inversion Formula in the Continuous case

We know

$$B_{n+1}'(x) = (n+1)B_n(x). \tag{6.23}$$

Now,

$$B'_{n+1}(x) = (n+1)B_n(x),$$
$$B_n(x) = \frac{1}{n+1}B'_{n+1}(x),$$

Integrating both sides from limits y to $y+1$, we get,

$$\int_y^{y+1} B_n(x)\,dx = \frac{1}{n+1}\int_y^{y+1} B'_{n+1}(x)\,dx$$
$$= \frac{1}{n+1}\left[B_{n+1}(y+1) - B_{n+1}(y)\right]. \tag{6.24}$$

using Eq.[(6.18)] we can write from Eq.[6.24] that,

$$\int_y^{y+1} B_n(x)dx = \frac{1}{(n+1)}(n+1)y^n,$$
$$= y^n. \tag{6.25}$$

So we can finally write,

$$x^n = \int_x^{x+1} B_n(t)dt. \tag{6.26}$$

This is known as the inversion formula for the continuous case.

6.6.6 Translation formula for Bernoulli Polynomials

The translation of the Bernoulli polynomial (meaning $x \to x+y$) is given by,

$$B_n(x+y) = \sum_{k=0}^{n} {}^nC_k\ B_k(x)y^{n-k}. \tag{6.27}$$

Proof: Let us start from the equation Eq.[6.15],

$$B_n(t) = \sum_{k=0}^{n} {}^nC_k\ B_k t^{n-k}, \tag{6.28}$$
$$B_n(x+y) = \sum_{l=0}^{n} {}^nC_l\ B_l(x+y)^{n-l}. \tag{6.29}$$

This boils down to,

$$B_n(x+y) = \sum_{l=0}^{n} {}^nC_l\ B_l \sum_{k=0}^{n-l} {}^{n-l}C_k\ y^{n-k-l}x^k, \quad [\text{since } (x+y)^n = \sum_{k=0}^{n} {}^nC_k\ y^{n-k}x^k].$$

$$
\begin{aligned}
B_n(x+y) &= \sum_{l=0}^{n} {}^nC_l \; B_l \sum_{k=l}^{n} {}^{n-l}C_{k-l} \; y^{n-k}x^{k-l}, && \text{[Replacing } k \text{ by } (k-l)\text{]} \\
&= \sum_{0\leqslant l\leqslant k\leqslant n} {}^nC_l \; {}^{n-l}C_{k-l} \; B_l \, y^{n-k}x^{k-l}, \\
&= \sum_{0\leqslant l\leqslant k\leqslant n} {}^nC_k \, {}^kC_l B_l \, y^{n-k}x^{k-l}, && \text{[since } {}^nC_l \, {}^{n-l}C_{k-l} = {}^nC_k \; {}^kC_l, \\
& && \text{by a combinatorial identity]} \\
&= \sum_{k=0}^{n} {}^nC_k \left[\sum_{l=0}^{k} {}^kC_l B_l x^{k-l} \right] y^{n-k}, \\
&= \sum_{k=0}^{n} {}^nC_k B_k(x) y^{n-k}, && \text{[since } B_k(x) = \sum_{l=0}^{k} {}^kC_l B_l x^{k-l}\text{]} \\
B_n(x+y) &= \sum_{k=0}^{n} {}^nC_k B_k(x) y^{n-k}. && \textit{(proved)}
\end{aligned}
$$

6.6.7 Inversion Formula in the discrete case

We can start from Eq.[6.18],

$$
\begin{aligned}
B_n(x+1) - B_n(x) &= nx^{n-1}, \\
B_{n+1}(x+1) - B_{n+1}(x) &= (n+1)x^n,
\end{aligned}
$$

$$
\begin{aligned}
x^n &= \frac{1}{n+1}[B_{n+1}(x+1) - B_{n+1}(x)], \\
&= \frac{1}{n+1}\left[\sum_{k=0}^{n+1} {}^{n+1}C_k \; B_k(x) - B_{n+1}(x) \right], \text{(from Equation (31), putting } y=1) \\
&= \frac{1}{n+1}\left[\sum_{k=0}^{n} {}^{n+1}C_k \; B_k(x) + {}^{n+1}C_{n+1} \; B_{n+1}(x) - B_{n+1}(x) \right], \\
&= \frac{1}{n+1}\left[\sum_{k=0}^{n} \frac{(n+1)!}{(n+1-k)!k!} B_k(x) \right], \\
&= \frac{1}{n+1}\left[\sum_{k=0}^{n} \frac{(n+1)n!}{(n+1-k)(n-k)!k!} B_k(x) \right], \\
&= \frac{1}{n+1}\left[\sum_{k=0}^{n} \frac{(n+1)}{(n+1-k)} {}^nC_k \; B_k(x) \right].
\end{aligned}
$$

$$
x^n = \sum_{k=0}^{n} \frac{1}{(n+1-k)} {}^nC_k \; B_k(x). \tag{6.30}
$$

This is the inversion relation in the discrete case.

6.7 Fibonacci polynomial

Till now, we have discussed the Bernoulli numbers, Bernoulli polynomials, and some properties of them. Let us discuss something about the Fibonacci polynomial now, and we are also going to discuss the relation between the Fibonacci and Bernoulli polynomials in the next sections. The motivation for this study lies in the fact that we are interested in studying how the relation between the two number series, as well as polynomials, affects the entanglement between qubits arranged in a spin chain system. We know that the Fibonacci polynomials can be given by,

$$\begin{aligned}
F_1(x) &= 1,\\
F_2(x) &= x,\\
F_3(x) &= x^2+1,\\
F_4(x) &= x^3+2x,\\
F_5(x) &= x^4+3x^2+1,\\
F_6(x) &= x^5+4x^3+3x,\\
&\vdots
\end{aligned}$$

If we want to get a general expression to express the Fibonacci polynomials, then we have to search for a pattern. For that purpose, if we carefully observe the polynomials, then they can also be expressed as follows.

$$\begin{aligned}
F_1(x) &= {}^0C_0,\\
F_2(x) &= {}^1C_0\ x,\\
F_3(x) &= {}^2C_0\ x^2+{}^1C_1,\\
F_4(x) &= {}^3C_0\ x^3+{}^2C_1x,\\
F_5(x) &= {}^4C_0x^4+{}^3C_1x^2+{}^2C_2,\\
&\vdots
\end{aligned}$$

by observing the pattern we can write the generalized expression as follows

$$F_{n+1}(x) = {}^nC_0\ x^n+{}^{n-1}C_1\ x^{n-2}+...{}^{n-\lfloor\frac{n}{2}\rfloor}C_{\lfloor\frac{n}{2}\rfloor}x^{n-2\lfloor\frac{n}{2}\rfloor}.$$

As we are observing that $F_1(x)$ and $F_2(x)$ consist of one term, $F_3(x)$ and $F_4(x)$ consists of two terms, $F_5(x)$ and $F_6(x)$ consists of three terms and so on, also the binomial coefficient is

Therefore,

$$F_{n+1}(x) = \sum_{i=0}^{\lfloor \frac{n}{2} \rfloor} {}^{n-i}C_i \; x^{n-2i} \tag{6.31}$$

Here $\lfloor \frac{n}{2} \rfloor$ means floor function of $\frac{n}{2}$.

What are Floor Function and Ceiling Functions?

Floor function is represented as $\lfloor \; \rfloor$. It is defined as rounding a number up to the nearest smallest integer. This means that if I write $\lfloor 1.5 \rfloor$, it will give me 1, i.e., the nearest lowest possible integer. Similarly, if I write $\lfloor 23.8 \rfloor$, it will simply give me 23. Now consider $\lfloor -2.3 \rfloor$; what will be the result? In that case, it will give you the nearest lowest possible integer, which is -3, not -2.

Similarly, we can define the ceiling function as rounding a number up to the nearest largest integer. If I write $\lfloor 1.5 \rfloor$, it will give me 2. Similarly, if I write $\lfloor -2.3 \rfloor$ it will give -2 not -3.

Now from Eq.[6.31] we can write replacing n as $2N$ to get the general expression of the Fibonacci polynomial,

$$F_{2N+1}(x) = \sum_{m=0}^{N} {}^{2N-m}C_m \; x^{2N-2m} \tag{6.32}$$

6.8 Relation between Fibonacci and Bernoulli polynomials

We have discussed the Bernoulli polynomials and some of their properties, as well as the Fibonacci polynomial. In this section, we are aiming to relate the two polynomials by utilizing their properties mentioned earlier. With this motivation in mind, we can

write from Eq.[6.30],

$$x^n = \sum_{k=0}^{n} \frac{1}{(n+1-k)} {}^nC_k \; B_k(x).$$

$$x^{2N-2m} = \sum_{k=0}^{2N-2m} \frac{1}{(2N-2m+1-k)} {}^{2N-2m}C_k \; B_k(x).$$

Again from Eq.[6.32] we have come to know that

$$F_{2N+1}(x) = \sum_{m=0}^{N} {}^{2N-m}C_m \; x^{2N-2m}.$$

Therefore, we can write,

$$\begin{aligned}
F_{2N+1}(x) &= \sum_{m=0}^{N} {}^{2N-m}C_m \; x^{2N-2m}, \\
&= \sum_{m=0}^{N} {}^{2N-m}C_m \sum_{k=0}^{2N-2m} \frac{1}{(2N-2m+1-k)} {}^{2N-2m}C_k \; B_k(x), \\
&= \sum_{k=0}^{2N} B_k(x) \sum_{m=0}^{\frac{2N-k}{2}} {}^{2N-m}C_m \; {}^{2N-2m}C_k \frac{1}{2N-2m-k+1}, \\
&= \sum_{k=0}^{2N} C_{k,N} \; B_k(x). \\
\text{where, } C_{k,N} &= \sum_{m=0}^{\frac{2N-k}{2}} {}^{2N-m}C_m \; {}^{2N-2m}C_k \frac{1}{2N-2m-k+1}
\end{aligned}$$

So in a compact way, the relation between the Fibonacci and Bernoulli polynomials can be expressed as,

$$\boxed{F_{2N+1}(x) = \sum_{k=0}^{2N} C_{k,N} \; B_k(x)} \; . \tag{6.33}$$

6.9 Chapter Summary

We aim to study how the relationship between the two number series and polynomials (Fibonacci and Bernoulli) affects the entanglement between the qubits when we consider a random spin chain system involving these number series as the interaction strengths between the nearest neighbour qubit pairs. With this motivation in mind, in this chapter, we introduced the Bernoulli numbers to help the reader to get an

idea of them. We also discussed the generation of the Bernoulli numbers from the generating functions, as well as the recurrence relation, both analytically and numerically. Then we pointed out some of the interesting properties of Bernoulli numbers and polynomials by utilizing which we end up with an interesting relations that exist between the Bernoulli and the Fibonacci polynomials.

Chapter 7

One dimensional Ising-like Model with Bernoulli type of interaction and Entanglement Studies

---

7.1 Introduction

In the previous chapter, we have come to know that a relation exists between Fibonacci and Bernoulli polynomials. In this chapter, we are going to see how the entanglement between the qubits is when we consider the interaction energies between the qubits to be varied as Bernoulli numbers, and we are interested in seeing whether the relation between the Fibonacci and Bernoulli numbers is reflected in their entanglement. For that, we first need to construct the Hamiltonian of an Ising-like system. The Hamiltonian corresponding to the system can be given as follows.

$$\mathcal{H} = \sum_i J_i \sigma_i^z \sigma_{i+1}^z + B \sum_i \sigma_i^x \tag{7.1}$$

We had considered the interaction between the neighboring spins to be along the Z-direction and the external magnetic field B applied along the X-direction, i.e., transverse to the spin chain system. Here, J_i's are the interaction energies between the neighboring spins and they are chosen from the successive terms of the Bernoulli series $i.e. J_i \in \{B_i\}$.[1]

[1]Here, B is used to designate the magnetic field, and $\{B_i\}$ represents the set of Bernoulli numbers throughout the thesis.

7.2 Study of Eigenvalue Spectrum for two different Cases

We can consider a one-dimensional spin chain system consisting of qubits in both open chain system configuration and periodic chain system configuration, where the interaction energies are varied like succeeding Bernoulli numbers.

Case-I: Open Chain System

The Hamiltonian of the system for this case can be given by:

$$\mathcal{H} = \sum_{i=1}^{N-1} J_i \sigma_i^z \sigma_{i+1}^z + B \sum_{i=1}^{N} \sigma_i^x \tag{7.2}$$

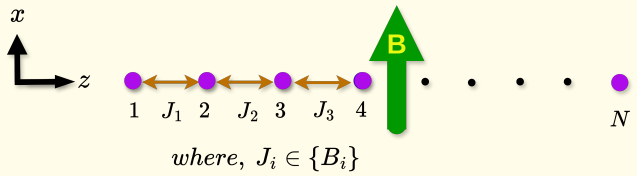


Figure 7.1: Open chain system

Case-II: Periodic Chain System

The Hamiltonian of the system for this case can be given by:

$$\mathcal{H} = \sum_{i=1}^{N} J_i \sigma_i^z \sigma_{i+1}^z + B \sum_{i=1}^{N} \sigma_i^x \tag{7.3}$$

where $\sigma_{N+1} = \sigma_1$.

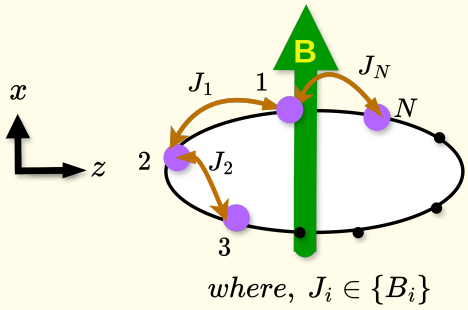


Figure 7.2: Periodic chain system

Now we are going to get the energy eigenvalues and the variation of energy eigenvalue with the magnetic field by constructing the Hamiltonian of the system numerically, without using the Tensor product approach by simple bit test. The approach is already described in detail in Chapter 1 while solving our usual Ising Model. Since we have two separate cases, we are going to discuss these two in two different subsections as follows.

7.2.1 Case(1) Open Chain System

We have constructed the following Fortran code to obtain the system's eigenvalues and study how they vary with the change in the magnetic field for an open chain system.

```
program Isinggen_with_Bernoulli_open_chain
!J is varied like the Bernoulli series, and B is also varied
!Eigenvalue open chain
implicit none
integer :: n, i, num, a, x, y, k, t, c, differing_bits
real*8 :: B
integer:: m
character(len=:), allocatable :: state(:)
integer, allocatable:: decimal(:)
Integer, allocatable :: psi(:)
double precision, allocatable :: p(:)
double precision, allocatable :: vec1(:), res1(:), vit(:)
integer :: j
real*8, allocatable :: S1(:,:), S2(:,:), H(:,:), H1(:,:), S3(:,:)

! Parameters for LAPACK DSYEV
integer:: lda, info, lwork
real*8, allocatable :: w(:), work(:)
character*1 :: JOBZ, UPLO

! Input values
print*, "Enter the number of bits you want:"
read*, n
```

```
24
25 ! Calculate number of states
26 num = 2**n
27 m = n
28 ! Allocate arrays after determining the value of num
29 allocate(character(len=n) :: state(num))
30 allocate(integer :: decimal(num))
31 allocate(psi(n))
32 allocate(S1(num, num))
33 allocate(S2(num, num))
34 allocate(S3(num, num))
35 allocate(H(num, num))
36 allocate(H1(num, num))
37 allocate(p(0:m))
38 allocate(rho(num,num))
39 allocate(site(0:d-1))
40 allocate(vec1(num))
41 allocate(vit(0:num-1))
42
43 ! LAPACK parameters
44 lda = num
45 lwork = 3 * num - 1
46 allocate(real(8) :: w(lda), work(lwork))
47
48 ! Printing the possible states
49 !print*, "The states are:"
50 do i = 0, num - 1
51     call integer_binary(i, state(i + 1), n)  ! Adjusting index for
    1-based
52     !print *, state(i + 1)
53 end do
54
55 ! Convert binary states to decimal
56 do i = 1, num
57     decimal(i) = btod(state(i), n)
58 end do
59
60 call Bernoulli_number(m, p)
```

```
write(9,*) p
! Perform calculations based on the decimal values for S1
do i = 1, num
    do j = 1, num
        call single_digit(state(i), psi)
        if (decimal(i) /= decimal(j)) then
            S1(i, j) = 0.0
        else
            S1(i, j) = 0.0
            do k = 2, n
                S1(i, j) = S1(i, j) + p(k-2)*(1.0 - 2.0 * psi(k-1))
   * (1.0 - 2.0 * psi(k))
            end do
        end if
    end do
end do

! Construct the S3 matrix (for sigma_x )
S3 = 0.0
do i = 1, num
    do j = 1, num
    ! Initialize differing_bits counter
        differing_bits = 0
        do k = 1, n
            ! Compare the k-th bit of the i-th and j-th states
            if (btest(decimal(i), k-1) .neqv. btest(decimal(j), k-1)
   ) then
                differing_bits = differing_bits + 1
            end if
        end do
    ! If exactly one bit differs, set S2(i, j) = 1.0
        if (differing_bits == 1) then
            S3(i, j) = 1.0
        else
            S3(i, j) = 0.0
        end if
    end do
end do
```

```
97
98  !Total Hamiltonian
99  !write(722,*) "H Array:"
100 do y = 0, 100, 1
101     B = y * 0.01
102     H = S1 + B*S3
103     !write(722,*)   H
104     ! Eigenvalue and eigenvector determination using DSYEV
105     JOBZ = 'V'
106     UPLO = 'U'
107     call DSYEV(JOBZ, UPLO, num, H, lda, w, work, lwork, info)
108     if (info == 0) then
109         do a = 1, lda
110            write(1225,*) B, w(a) !plot for magfield vs eigenvalue
111         end do
112     else
113         write(1225,*) 'Error in DSYEV, INFO=', info
114     end if
115     vec1 = H(:, 1)
116     do i = 1, num
117         vit(i-1) = vec1(i)
118     end do
119 end do
120
121 deallocate(state)
122 deallocate(decimal)
123 deallocate(psi)
124 deallocate(S1)
125 deallocate(S2)
126 deallocate(S3)
127 deallocate(H)
128 deallocate(H1)
129 deallocate(p)
130 deallocate(rho)
131 deallocate(vit)
132 deallocate(vec1)
133
134 contains
```

```
135     function btod(state, n) result(decimal)
136         implicit none
137         integer, intent(in) :: n
138         character(len=n) :: state
139         integer :: decimal, i
140         decimal = 0
141         do i = 1, n
142             if (state(i:i) == '1') then
143                 decimal = decimal + 2**(n-i)
144             end if
145         end do
146     end function btod
147 
148     subroutine integer_binary(num, binary_string, n)
149         implicit none
150         integer :: num
151         integer, intent(in) :: n
152         integer :: i, temp_num
153         character(len=n), intent(out) :: binary_string
154         temp_num = num
155         binary_string = repeat('0', n)
156         do i = 0, n - 1
157             if (mod(temp_num, 2) == 1) then
158                 binary_string(n-i:n-i) = '1'
159             else
160                 binary_string(n-i:n-i) = '0'
161             end if
162             temp_num = temp_num / 2
163         end do
164     end subroutine integer_binary
165 
166     subroutine single_digit(A, psi)
167         implicit none
168         character(len=*), intent(in) :: A
169         integer, intent(out) :: psi(:)
170         integer :: i
171         do i = 1, len(A)
172             if (A(i:i) == '0') then
```

```
                psi(i) = 0
            else if (A(i:i) == '1') then
                psi(i) = 1
            else
                print *, "Invalid input in single_digit subroutine"
            end if
        end do
    end subroutine single_digit

    subroutine Bernoulli_number(n, B)
    implicit none
    double precision :: B(0:n)
    integer :: n, i, j, k
    double precision :: sum


    if (n <= 0) then
        print*, "Number of terms must be greater than 0."
        stop
    end if

    ! Initialization of the first term for the series
    B(0) = 1.0d0  ! B(0) = 1

    ! Calculate the Bernoulli numbers
    do j = 1, n
        sum = 0.0d0
        do k = 0, j - 1
            sum = sum + (binomial_coefficient(j + 1, k)  * B(k))
        end do
        B(j) = -sum / (j + 1.0d0)
    end do

    ! Print the Bernoulli series
    !print*, "Bernoulli series:"
    do i = 0, n
        write(92,*) i, B(i)
    end do
```

```
211
212     end subroutine Bernoulli_number
213
214     ! Function to calculate binomial coefficient
215     function binomial_coefficient(n, k) result(coefficient)
216         implicit none
217         integer, intent(in) :: n, k
218         double precision :: coefficient
219         integer :: i
220         coefficient = 1.0d0
221         if (k > 0 .and. k < n) then
222             do i = 1, k
223                 coefficient = coefficient * (n - i + 1) / i
224             end do
225         else if (k == 0 .or. k == n) then
226             coefficient = 1.0d0
227         else
228             coefficient = 0.0d0
229         end if
230     end function binomial_coefficient
231 end program Isinggen_with_bernoulli_open_chain
```

We have constructed the code such that the $S1$ matrix (mentioned in the above code) will calculate the first summation term of the Eq.[7.2], and the $S3$ matrix (mentioned in the above code) will calculate the second summation term containing σ^x. Then, we have constructed the Hamiltonian by varying the magnetic field from 0 to 1 unit in steps of 0.01 units. The variation of energy eigenvalue with the change in the magnetic field can be seen from the Fig.[7.3] where we have obtained the result considering a three-qubit open chain system. The variation of the energy eigenvalues is non-linear, as observed, and it is quite obvious as already described. The Hamiltonian of the system comprises the σ^z and σ^x, and since we know that σ^z and σ^x are non-commuting, which implies that we can not have an eigenfunction that can diagonalize both at the same time. This has already been discussed in Chapter 1.

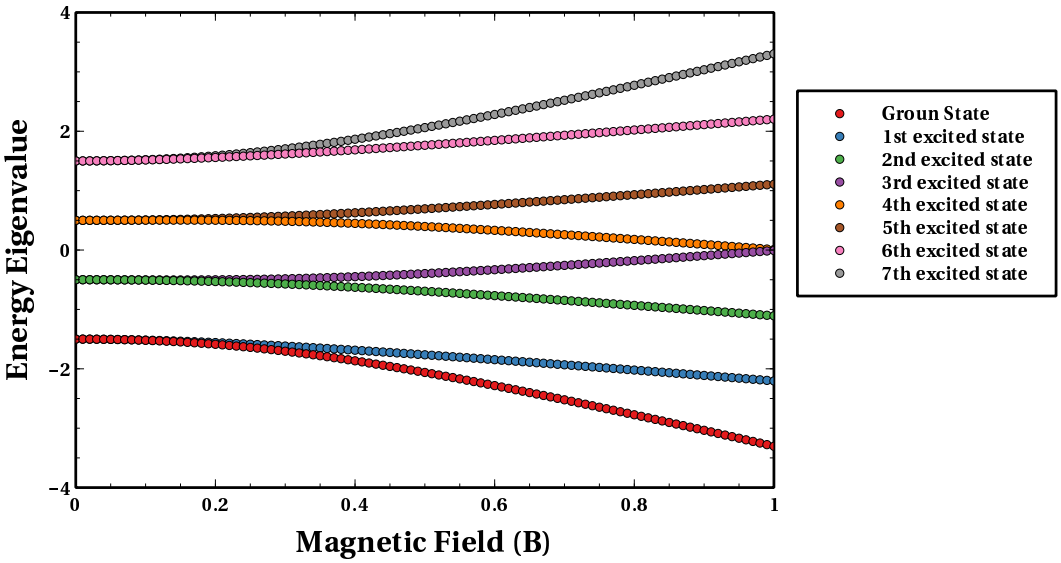


Figure 7.3: Energy eigenvalue spectrum with variation of magnetic field for 3 qubits open chain system

7.2.2 Case(2) Periodic Chain System

Here we have constructed the following Fortran code to obtain the system's eigenvalues and study how they vary with the change in the magnetic field for a periodic chain system imposing the periodic or wrap-up condition, which is $\sigma_{N+1} \equiv \sigma_1$.

```
program Isinggen_with_Bernoulli_periodic_chain
!J is varied like Bernoulli series and B is also varied
!Periodic chain
implicit none
integer :: n, i, num, a, x, y, k, t, c, differing_bits
real*8 :: B
integer :: m
character(len=:), allocatable :: state(:)
integer, allocatable :: decimal(:)
integer, allocatable :: psi(:)
double precision, allocatable :: p(:)
double precision, allocatable :: vec1(:), res1(:), vit(:)
integer :: j
real*8, allocatable :: S1(:,:), H(:,:), H1(:,:), S3(:,:)

```

```
! Parameters for LAPACK DSYEV
integer :: lda, info, lwork
real*8, allocatable :: w(:), work(:)
character*1 :: JOBZ, UPLO

! Input values
print*, "Enter the number of bits you want:"
read*, n

! Calculate number of states
num = 2**n
m = n

! Allocate arrays after determining the value of num
allocate(character(len=n) :: state(num))
allocate(integer :: decimal(num))
allocate(psi(n))
allocate(S1(num, num))
allocate(S2(num, num))
allocate(S3(num, num))
allocate(H(num, num))
allocate(H1(num, num))
allocate(p(0:m))
allocate(rho(num,num))
allocate(site(0:d-1))
allocate(vec1(num))
allocate(vit(0:num-1))

! LAPACK parameters
lda = num
lwork = 3 * num - 1
allocate(real(8) :: w(lda), work(lwork))

! Printing the possible states
print*, "The states are:"
do i = 0, num - 1
    call integer_binary(i, state(i + 1), n)  ! Adjusting index for
   1-based
```

```
53      print *, state(i + 1)
54  end do
55
56  !write(722,*) "Decimal equivalents of the states"
57  ! Convert binary states to decimal
58  do i = 1, num
59      decimal(i) = btod(state(i), n)
60      !write(722,*) decimal(i)
61  end do
62
63  call Bernoulli_number(m, p)
64  write(9,*) p
65  ! Perform calculations based on the decimal values for S1
66  do i = 1, num
67      do j = 1, num
68          call single_digit(state(i), psi)
69          if (decimal(i) /= decimal(j)) then
70              S1(i, j) = 0.0
71          else
72          S1(i, j) = 0.0
73          do k = 2, n
74              S1(i, j) = S1(i, j) + p(k-2)*(1.0 - 2.0 * psi(k-1)) *
    (1.0 - 2.0 * psi(k))
75          end do
76          S1(i, j) = S1(i, j) + p(n-1)*(1.0 - 2.0 * psi(n)) * (1.0 -
    2.0 * psi(1))
77          end if
78      end do
79  end do
80
81  ! Construct the S3 matrix (for sigma_x )
82  S3 = 0.0
83  do i = 1, num
84      do j = 1, num
85      ! Initialize differing_bits counter
86      differing_bits = 0
87          do k = 1, n
88              ! Compare the k-th bit of the i-th and j-th states
```

```
            if (btest(decimal(i), k-1) .neqv. btest(decimal(j), k-1)
    ) then
                differing_bits = differing_bits + 1
            end if
        end do
        if (differing_bits == 1) then
            S3(i, j) = 1.0
        else
            S3(i, j) = 0.0
        end if
    end do
end do

! Total Hamiltonian
!write(722,*) "H Array:"
do y = 0, 100, 1
    B = y * 0.01
    H = S1 + B*S3
    ! Eigenvalue and eigenvector determination using DSYEV
    JOBZ = 'V'
    UPLO = 'U'
    call DSYEV(JOBZ, UPLO, num, H, lda, w, work, lwork, info)
    if (info == 0) then
        do a = 1, lda
            write(1224,*) B, w(a) !plot for magfield vs eigenvalue
        end do
    else
        write(1223,*) 'Error in DSYEV, INFO=', info
    end if
    vec1 = H(:, 1)
    do i = 1, num
        vit(i-1) = vec1(i)
    end do
end do

deallocate(state)
deallocate(decimal)
deallocate(psi)
```

```
126 deallocate(S1)
127 deallocate(S2)
128 deallocate(S3)
129 deallocate(H)
130 deallocate(H1)
131 deallocate(p)
132 deallocate(rho)
133 deallocate(vit)
134 deallocate(vec1)
135
136 contains
137
138     function btod(state, n) result(decimal)
139         implicit none
140         integer, intent(in) :: n
141         character(len=n) :: state
142         integer :: decimal, i
143         decimal = 0
144         do i = 1, n
145             if (state(i:i) == '1') then
146                 decimal = decimal + 2**(n-i)
147             end if
148         end do
149     end function btod
150
151     subroutine integer_binary(num, binary_string, n)
152         implicit none
153         integer :: num
154         integer, intent(in) :: n
155         integer :: i, temp_num
156         character(len=n), intent(out) :: binary_string
157         temp_num = num
158         binary_string = repeat('0', n)
159         do i = 0, n - 1
160             if (mod(temp_num, 2) == 1) then
161                 binary_string(n-i:n-i) = '1'
162             else
163                 binary_string(n-i:n-i) = '0'
```

```
            end if
            temp_num = temp_num / 2
        end do
    end subroutine integer_binary

    subroutine single_digit(A, psi)
        implicit none
        character(len=*), intent(in) :: A
        integer, intent(out) :: psi(:)
        integer :: i
        do i = 1, len(A)
            if (A(i:i) == '0') then
                psi(i) = 0
            else if (A(i:i) == '1') then
                psi(i) = 1
            else
                print *, "Invalid input in single_digit subroutine"
            end if
        end do
    end subroutine single_digit

    subroutine Bernoulli_number(n, B)
    implicit none
    double precision :: B(0:n)
    integer :: n, i, j, k
    double precision :: sum


    if (n <= 0) then
        print*, "Number of terms must be greater than 0."
        stop
    end if

    ! Initialization of the first term for the series
    B(0) = 1.0d0  ! B(0) = 1

    ! Calculate the Bernoulli numbers
    do j = 1, n
```

```
202         sum = 0.0d0
203         do k = 0, j - 1
204             sum = sum + (binomial_coefficient(j + 1, k)  * B(k))
205         end do
206         B(j) = -sum / (j + 1.0d0)
207     end do
208 
209     ! Print the Bernoulli series
210     !print*, "Bernoulli series:"
211     do i = 0, n
212         write(92,*) i, B(i)
213     end do
214 
215     end subroutine Bernoulli_number
216 
217     ! Function to calculate binomial coefficient
218     function binomial_coefficient(n, k) result(coefficient)
219         implicit none
220         integer, intent(in) :: n, k
221         double precision :: coefficient
222         integer :: i
223         coefficient = 1.0d0
224         if (k > 0 .and. k < n) then
225             do i = 1, k
226                 coefficient = coefficient * (n - i + 1) / i
227             end do
228         else if (k == 0 .or. k == n) then
229             coefficient = 1.0d0
230         else
231             coefficient = 0.0d0
232         end if
233     end function binomial_coefficient
234 end program Isinggen_with_bernoulli_periodic_chain
```

Using this code, we have studied the variations of the energy eigenvalues with the change in the magnetic field from 0 to 1 unit in steps of 0.01 units for a three-qubit periodic chain system. We have observed that in this case, the variation is also not

linear due to the same reason as mentioned in the open chain system, i.e., we can not find a single eigenfunction that can diagonalize both σ^z and σ^x. The variations are shown in Fig.[7.4].

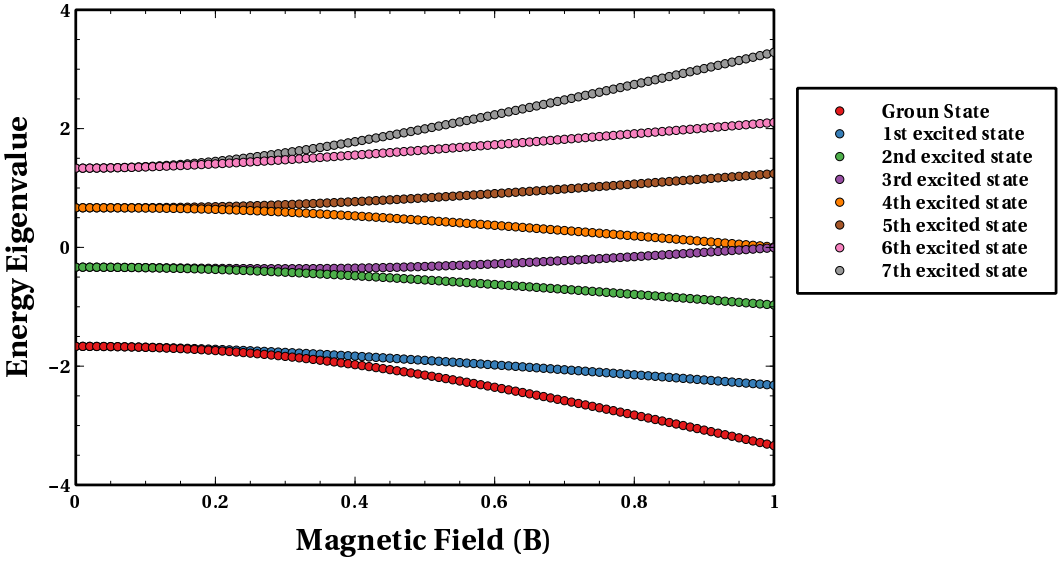


Figure 7.4: Energy eigenvalue spectrum with variation of magnetic field for 3 qubits periodic chain system

7.3 Entanglement Study

We are going to study the entanglement between different qubit pairs by taking into account the two tools -Concurrence and von Neumann Entropy, which we had already discussed in Chapter 4.

7.3.1 Study of Concurrence

In the subsequent sections, let us discuss the variation of the concurrences with the external magnetic field for two separate cases.

7.3.1.1 Case(1) Open Chain System

We have already constructed the Fortran code to construct the Hamiltonian of the system under study numerically. We also found the system's energy eigenvalues with variations in the applied magnetic field. Now, by modifying our code, we can obtain the system's ground eigenstate and calculate the concurrences for different qubit pairs.

```
program Isinggen_with_Bernoulli_open_chain_concurrence
!J is varied like the Bernoulli series, and B is also varied
!Concurrence calculation open chain
implicit none
integer :: n, i, num, a, x, y, k, t, c, differing_bits
real*8 :: B
real:: con
integer :: m
integer, parameter:: d = 2
character(len=:), allocatable :: state(:)
integer, allocatable :: decimal(:)
integer, allocatable :: psi(:)
double precision, allocatable :: p(:)
double precision, allocatable :: vec1(:), res1(:), vit(:)
integer :: j
real*8, allocatable :: S1(:,:), S2(:,:), H(:,:), H1(:,:), S3(:,:),
   rho(:,:)
integer, allocatable:: site(:)

! Parameters for LAPACK DSYEV
integer:: lda, info, lwork
real*8, allocatable :: w(:), work(:)
character*1 :: JOBZ, UPLO

! Input values
print*, "Enter the number of bits you want:"
read*, n

! Calculate number of states
```

```
num = 2**n
m = n
! Allocate arrays after determining the value of num
allocate(character(len=n) :: state(num))
allocate(integer :: decimal(num))
allocate(psi(n))
allocate(S1(num, num))
allocate(S2(num, num))
allocate(S3(num, num))
allocate(H(num, num))
allocate(H1(num, num))
allocate(p(0:m))
allocate(rho(num,num))
allocate(site(0:d-1))
allocate(vec1(num))
allocate(vit(0:num-1))

! LAPACK parameters
lda = num
lwork = 3 * num - 1
allocate(real(8) :: w(lda), work(lwork))

! Printing the possible states
!print*, "The states are:"
do i = 0, num - 1
    call integer_binary(i, state(i + 1), n)  ! Adjusting index for
   1-based
    !print *, state(i + 1)
end do

! Convert binary states to decimal
do i = 1, num
    decimal(i) = btod(state(i), n)
end do

call Bernoulli_number(m, p)
write(9,*) p
! Perform calculations based on the decimal values for S1
```

```
do i = 1, num
    do j = 1, num
        call single_digit(state(i), psi)
        if (decimal(i) /= decimal(j)) then
            S1(i, j) = 0.0
        else
            S1(i, j) = 0.0
            do k = 2, n
                S1(i, j) = S1(i, j) + p(k-2)*(1.0 - 2.0 * psi(k-1))
   * (1.0 - 2.0 * psi(k))
            end do
        end if
    end do
end do

! Construct the S3 matrix (for sigma_x )
S3 = 0.0
do i = 1, num
    do j = 1, num
    ! Initialize differing_bits counter
        differing_bits = 0
        do k = 1, n
            ! Compare the k-th bit of the i-th and j-th states
            if (btest(decimal(i), k-1) .neqv. btest(decimal(j), k-1)
   ) then
                differing_bits = differing_bits + 1
            end if
        end do
    ! If exactly one bit differs, set S2(i, j) = 1.0
        if (differing_bits == 1) then
            S3(i, j) = 1.0
        else
            S3(i, j) = 0.0
        end if
    end do
end do

!Total Hamiltonian
```

```
!write(722,*) "H Array:"
do y = 0, 100, 1
    B = y * 0.01
    H = S1 + B*S3
    !write(722,*)   H
    ! Eigenvalue and eigenvector determination using DSYEV
    JOBZ = 'V'
    UPLO = 'U'
    call DSYEV(JOBZ, UPLO, num, H, lda, w, work, lwork, info)
    if (info == 0) then
        do a = 1, lda
           write(1225,*) B, w(a) !plot for magfield vs eigenvalue
        end do
    else
        write(1225,*) 'Error in DSYEV, INFO=', info
    end if
    vec1 = H(:, 1)
    do i = 1, num
        vit(i-1) = vec1(i)
    end do
    !Study of Concurrence
    do i = 0, n-1
        do j = i+1, n-1
            if( i /= (n-1)) then
                call CONVR(n,1,2,vit,con)
                write(941,*) B, con
            else
                EXIT
            end if
        end do
    end do
end do

deallocate(state)
deallocate(decimal)
deallocate(psi)
deallocate(S1)
deallocate(S2)
```

```
deallocate(S3)
deallocate(H)
deallocate(H1)
deallocate(p)
deallocate(rho)
deallocate(vit)
deallocate(vec1)

contains
function btod(state, n) result(decimal)
    implicit none
    integer, intent(in) :: n
    character(len=n) :: state
    integer :: decimal, i
    decimal = 0
    do i = 1, n
        if (state(i:i) == '1') then
            decimal = decimal + 2**(n-i)
        end if
    end do
end function btod

subroutine integer_binary(num, binary_string, n)
    implicit none
    integer :: num
    integer, intent(in) :: n
    integer :: i, temp_num
    character(len=n), intent(out) :: binary_string
    temp_num = num
    binary_string = repeat('0', n)
    do i = 0, n - 1
        if (mod(temp_num, 2) == 1) then
            binary_string(n-i:n-i) = '1'
        else
        binary_string(n-i:n-i) = '0'
        end if
        temp_num = temp_num / 2
    end do
```

```
178 end subroutine integer_binary
179
180 subroutine single_digit(A, psi)
181     implicit none
182     character(len=*), intent(in) :: A
183     integer, intent(out) :: psi(:)
184     integer:: i
185     do i = 1, len(A)
186         if (A(i:i) == '0') then
187             psi(i) = 0
188         else if (A(i:i) == '1') then
189               psi(i) = 1
190         else
191         print *, "Invalid input in single_digit subroutine"
192         end if
193     end do
194 end subroutine single_digit
195
196 subroutine Bernoulli_number(n, B)
197     implicit none
198     double precision :: B(0:n)
199     integer :: n, i, j, k
200     double precision:: sum
201     if (n <= 0) then
202         print*, "Number of terms must be greater than 0."
203         stop
204     end if
205
206     ! Initialization of the first term for the series
207     B(0) = 1.0d0  ! B(0) = 1
208
209     ! Calculate the Bernoulli numbers
210     do j = 1, n
211         sum = 0.0d0
212         do k = 0, j - 1
213             sum = sum + (binomial_coefficient(j + 1, k)  * B(k))
214         end do
215         B(j) = -sum / (j + 1.0d0)
```

```
    end do

    ! Print the Bernoulli series
    !print*, "Bernoulli series:"
    do i = 0, n
        write(92,*) i, B(i)
    end do

end subroutine Bernoulli_number

    ! Function to calculate binomial coefficient
    function binomial_coefficient(n, k) result(coefficient)
        implicit none
        integer, intent(in) :: n, k
        double precision :: coefficient
        integer :: i
        coefficient = 1.0d0
        if (k > 0 .and. k < n) then
            do i = 1, k
                coefficient = coefficient * (n - i + 1) / i
            end do
        else if (k == 0 .or. k == n) then
            coefficient = 1.0d0
        else
            coefficient = 0.0d0
        end if
    end function binomial_coefficient
subroutine CONVR(s,i0,j0,vin,con)
    implicit none
    integer::s,i0,j0 ,site(0:1),i
    real*8::vin(0:2**s-1),evs
    real*8,dimension(0:3,0:3)::rdm,MM1,MM2,MM3,MM4
    INTEGER::INFO, LDA, LWORK, NN,LDVL,LDVR
    parameter(NN=4,LDA=NN,LWORK=4*NN,LDVL=NN,LDVR=NN)
    real*8::WORK(0:LWORK-1),AA2(0:LDA-1,0:NN-1)
    real*8::VL(0:NN-1,0:NN-1),VR(0:NN-1,0:NN-1)
    real*8::WR(0:NN-1),WI(0:NN-1)
    real*8::ev(NN),c2,x11,const,sigmay(0:3,0:3)
```

```
254     real :: con
255     sigmay(0,:)=(/0.0,0.0,0.0,-1.0/)
256     sigmay(1,:)=(/0.0,0.0,1.0,0.0/)
257     sigmay(2,:)=(/0.0,1.0,0.0,0.0/)
258     sigmay(3,:)=(/-1.0,0.0,0.0,0.0/)
259     site(0)=i0
260     site(1)=j0
261     call PTRVR(s,2,site,vin,rdm)
262     write(3,*) rdm
263     MM1=matmul(rdm,sigmay)
264     MM2=matmul(rdm,sigmay)
265     MM3=matmul(MM1,MM2)
266     call dgeev('N', 'N', NN, MM3, LDA, WR, &
267         WI, VL, LDVL, VR, LDVR, WORK, LWORK, INFO)
268         do i=1,NN,1
269             if (WR(i-1) .lt. 0.0000000000001) then
270                 WR(i-1)=0.0d0
271             end if
272                 ev(i)=WR(i-1)
273         end do
274         do i=2,4
275             if(ev(i)>=ev(1))then
276                 evs=ev(i)
277                 ev(i)=ev(1)
278                 ev(1)=evs
279             end if
280         end do
281         do i=3,4
282             if(ev(i)>=ev(2))then
283                 evs=ev(i)
284                 ev(i)=ev(2)
285                 ev(2)=evs
286             end if
287         end do
288         do i=4,4
289             if(ev(i)>=ev(3))then
290                 evs=ev(i)
291                 ev(i)=ev(3)
```

```
292                 ev(3)=evs
293             end if
294         end do
295         c2=0
296         con=0
297         c2=dsqrt(ev(1))-dsqrt(ev(2))-dsqrt(ev(3))-dsqrt(ev(4))
298         ! print*, c2
299     if(c2>0)then
300         con=c2
301     else
302         con=0
303     end if
304 end subroutine
305 subroutine PTRVR(s, s1, site, vin, rdm)
306     implicit none
307     integer :: s, s1, s2
308     real*8 :: trace
309     real*8, dimension(0:2**s-1) :: vin
310     real*8, dimension(0:2**s1-1, 0:2**s1-1) :: rdm
311     integer :: ii, i1, a, i0, ia, j1, oo, k1, x1, y1, j, t1, t2, z1
312     integer, dimension(0:s1-1) :: site, site2
313     integer, dimension(0:s1-1) :: bin
314     integer, dimension(0:(2**(s-s1))*(2**s1)-1) :: ind
315     s2 = s - s1
316     x1 = 0
317     y1 = 0
318     ind = 0
319     do j1 = 2**s1-1, 0, -1
320         do ii = 0, 2**s-1
321             a = ii
322             do i1 = 0, s1-1
323                 i0 = site(i1)
324                 call DTOBONEBIT(a, ia, i0, s)
325                 site2(i1) = ia
326             end do
327             call DTOB(j1, bin, s1)
328             oo = 1
329             do k1 = 0, s1-1
```

```
330                 oo = oo * (bin(k1) - site2(k1))
331             end do
332             if (abs(oo) == 1) then
333                 ind(x1) = ii
334                 x1 = x1 + 1
335             end if
336         end do
337     end do
338
339     rdm = 0.0d0
340     do t1 = 0, 2**s1-1
341         do t2 = 0, 2**s1-1
342             do z1 = 0, 2**s2-1
343                 rdm(t1, t2) = rdm(t1, t2) + vin(ind((2**s2) * t1 +
    z1)) * vin(ind((2**s2) * t2 + z1))
344             end do
345         end do
346     end do
347 end subroutine
348
349 subroutine DTOBONEBIT(m, ia, i0, s)
350     implicit none
351     integer*4 :: s
352     integer :: tt(0:s-1)
353     integer :: m, ia, i0
354     call DTOB(m, tt, s)
355     ia = tt(i0)
356 end subroutine
357
358 subroutine DTOB(m, tt, s)
359     implicit none
360     integer :: s
361     integer, dimension(0:s-1) :: tt
362     integer :: m, k, a2
363     tt = 0
364     a2 = m
365     do k = 0, s-1
366         tt(s-k-1) = mod(a2, 2)
```

```
367             a2 = a2 / 2
368             if (a2 == 0) exit
369         end do
370     end subroutine
371 end program Isinggen_with_bernoulli_open_chain_concurrence
```

Implementing this code, we have studied the three-qubit open chain system by selecting n in the above program as 3. The variation of concurrences for different possible qubit pairs (for a three-qubit system, the concurrences will be obtained for ${}^3C_2 = 3$ combinations of qubit pairs which are (1,2),(1,3) & (2,3)) are plotted in the Fig.[7.5]. We can also use this program to study higher spin systems. Now, we can see that the concurrences are continuous and first increase up to a certain amount of magnetic field and then decrease, but the rate of increase is quite high compared to the rate of decrease. We can say that the entanglement between the qubit pair (1,2) is larger than compared of the other qubit pairs.

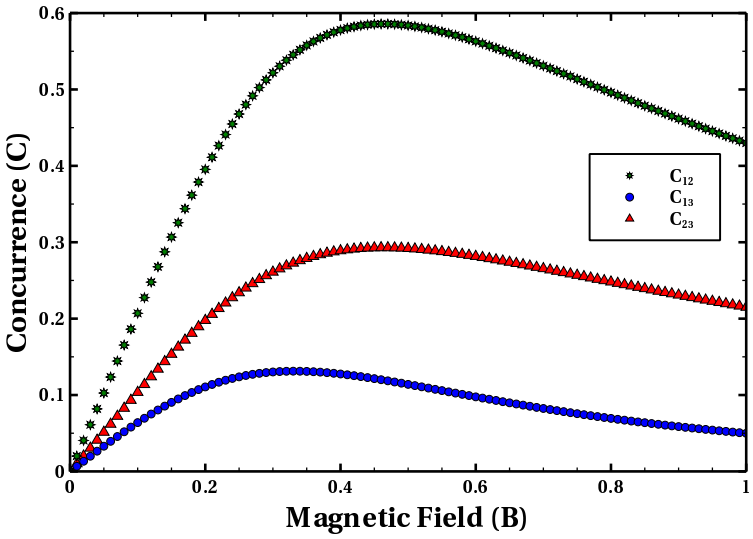


Figure 7.5: Variation of Concurrences with magnetic field for three-qubit open chain system.

If we carefully observe the variation, then we can see that the variation of concurrences is very similar to the variation of concurrences for the Inverse-Fibonacci Ising-like spin system, which we have already discussed in Chapter 4. We have plotted the variations of concurrences for these two spin chain systems simultaneously in

Fig.[7.6].

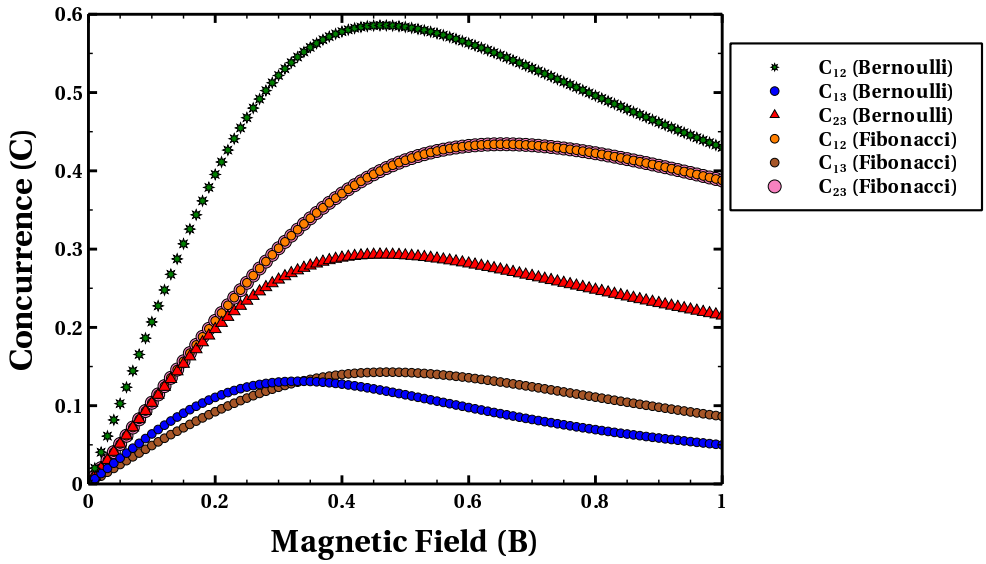


Figure 7.6: Variation of Concurrences with magnetic field for three-qubit open chain system (For both Inverse Fibonacci and Bernoulli type interaction).

7.3.1.2 Case(2) Periodic Chain System

For the periodic chain system, we can also construct the Fortran code as follows and calculate the concurrences.

```
program Isinggen_with_Bernoulli_periodic_chain_concurrence
!J is varied like the Bernoulli series, and B is also varied
!Concurrence calculation open chain
implicit none
integer :: n, i, num, a, x, y, k, t, c, differing_bits
real*8 :: B
real:: con
integer :: m
integer, parameter:: d = 2
character(len=:), allocatable :: state(:)
integer, allocatable :: decimal(:)
integer, allocatable :: psi(:)
double precision, allocatable :: p(:)
double precision, allocatable :: vec1(:), res1(:), vit(:)
integer :: j
```

```
real*8, allocatable :: S1(:,:), S2(:,:), H(:,:), H1(:,:), S3(:,:),
   rho(:,:)
integer, allocatable:: site(:)

! Parameters for LAPACK DSYEV
integer:: lda, info, lwork
real*8, allocatable :: w(:), work(:)
character*1 :: JOBZ, UPLO

! Input values
print*, "Enter the number of bits you want:"
read*, n

! Calculate number of states
num = 2**n
m = n
! Allocate arrays after determining the value of num
allocate(character(len=n) :: state(num))
allocate(integer :: decimal(num))
allocate(psi(n))
allocate(S1(num, num))
allocate(S2(num, num))
allocate(S3(num, num))
allocate(H(num, num))
allocate(H1(num, num))
allocate(p(0:m))
allocate(rho(num,num))
allocate(site(0:d-1))
allocate(vec1(num))
allocate(vit(0:num-1))

! LAPACK parameters
lda = num
lwork = 3 * num - 1
allocate(real(8) :: w(lda), work(lwork))

! Printing the possible states
!print*, "The states are:"
```

```
do i = 0, num - 1
    call integer_binary(i, state(i + 1), n)  ! Adjusting index for
   1-based
    !print *, state(i + 1)
end do

! Convert binary states to decimal
do i = 1, num
    decimal(i) = btod(state(i), n)
end do

call Bernoulli_number(m, p)
write(9,*) p
! Perform calculations based on the decimal values for S1
do i = 1, num
    do j = 1, num
        call single_digit(state(i), psi)
        if (decimal(i) /= decimal(j)) then
            S1(i, j) = 0.0
        else
        S1(i, j) = 0.0
        do k = 2, n
            S1(i, j) = S1(i, j) + p(k-2)*(1.0 - 2.0 * psi(k-1)) *
   (1.0 - 2.0 * psi(k))
        end do
        S1(i, j) = S1(i, j) + p(n-1)*(1.0 - 2.0 * psi(n)) * (1.0 -
   2.0 * psi(1))
        end if
    end do
end do

! Construct the S3 matrix (for sigma_x )
S3 = 0.0
do i = 1, num
    do j = 1, num
    ! Initialize differing_bits counter
    differing_bits = 0
        do k = 1, n
```

```
88              ! Compare the k-th bit of the i-th and j-th states
89              if (btest(decimal(i), k-1) .neqv. btest(decimal(j), k-1)
    ) then
90                  differing_bits = differing_bits + 1
91              end if
92          end do
93          if (differing_bits == 1) then
94              S3(i, j) = 1.0
95          else
96              S3(i, j) = 0.0
97          end if
98      end do
99  end do
100
101 !Total Hamiltonian
102 !write(722,*) "H Array:"
103 do y = 0, 100, 1
104     B = y * 0.01
105     H = S1 + B*S3
106     !write(722,*)   H
107     ! Eigenvalue and eigenvector determination using DSYEV
108     JOBZ = 'V'
109     UPLO = 'U'
110     call DSYEV(JOBZ, UPLO, num, H, lda, w, work, lwork, info)
111     if (info == 0) then
112         do a = 1, lda
113            write(1225,*) B, w(a) !plot for magfield vs eigenvalue
114         end do
115     else
116         write(1225,*) 'Error in DSYEV, INFO=', info
117     end if
118     vec1 = H(:, 1)
119     do i = 1, num
120         vit(i-1) = vec1(i)
121     end do
122     !Study of Concurrence
123     do i = 0, n-1
124         do j = i+1, n-1
```

```
            if( i /= (n-1)) then
                call CONVR(n,1,2,vit,con)
                write(942,*) B, con
            else
                EXIT
            end if
        end do
    end do
end do

deallocate(state)
deallocate(decimal)
deallocate(psi)
deallocate(S1)
deallocate(S2)
deallocate(S3)
deallocate(H)
deallocate(H1)
deallocate(p)
deallocate(rho)
deallocate(vit)
deallocate(vec1)

contains
    function btod(state, n) result(decimal)
        implicit none
        integer, intent(in) :: n
        character(len=n) :: state
        integer :: decimal, i
        decimal = 0
        do i = 1, n
            if (state(i:i) == '1') then
                decimal = decimal + 2**(n-i)
            end if
        end do
    end function btod

    subroutine integer_binary(num, binary_string, n)
```

```
        implicit none
        integer :: num
        integer, intent(in) :: n
        integer :: i, temp_num
        character(len=n), intent(out) :: binary_string
        temp_num = num
        binary_string = repeat('0', n)
        do i = 0, n - 1
            if (mod(temp_num, 2) == 1) then
                binary_string(n-i:n-i) = '1'
            else
                binary_string(n-i:n-i) = '0'
            end if
            temp_num = temp_num / 2
        end do
    end subroutine integer_binary

    subroutine single_digit(A, psi)
        implicit none
        character(len=*), intent(in) :: A
        integer, intent(out) :: psi(:)
        integer:: i
        do i = 1, len(A)
            if (A(i:i) == '0') then
                psi(i) = 0
            else if (A(i:i) == '1') then
                psi(i) = 1
            else
                print *, "Invalid input in single_digit subroutine"
            end if
        end do
    end subroutine single_digit

    subroutine Bernoulli_number(n, B)
    implicit none
    double precision :: B(0:n)
    integer :: n, i, j, k
    double precision:: sum
```

```
201
202
203     if (n <= 0) then
204         print*, "Number of terms must be greater than 0."
205         stop
206     end if
207
208     ! Initialization of the first term for the series
209     B(0) = 1.0d0  ! B(0) = 1
210
211     ! Calculate the Bernoulli numbers
212     do j = 1, n
213         sum = 0.0d0
214         do k = 0, j - 1
215             sum = sum + (binomial_coefficient(j + 1, k)  * B(k))
216         end do
217         B(j) = -sum / (j + 1.0d0)
218     end do
219
220     ! Print the Bernoulli series
221     !print*, "Bernoulli series:"
222     do i = 0, n
223         write(92,*) i, B(i)
224     end do
225
226     end subroutine Bernoulli_number
227
228     ! Function to calculate binomial coefficient
229     function binomial_coefficient(n, k) result(coefficient)
230         implicit none
231         integer, intent(in) :: n, k
232         double precision :: coefficient
233         integer :: i
234         coefficient = 1.0d0
235         if (k > 0 .and. k < n) then
236             do i = 1, k
237                 coefficient = coefficient * (n - i + 1) / i
238             end do
```

```
        else if (k == 0 .or. k == n) then
            coefficient = 1.0d0
        else
            coefficient = 0.0d0
        end if
    end function binomial_coefficient
    subroutine CONVR(s,i0,j0,vin,con)
        implicit none
        integer::s,i0,j0 ,site(0:1),i
        real*8::vin(0:2**s-1),evs
        real*8,dimension(0:3,0:3)::rdm,MM1,MM2,MM3,MM4
        INTEGER::INFO, LDA, LWORK, NN,LDVL,LDVR
        parameter(NN=4,LDA=NN,LWORK=4*NN,LDVL=NN,LDVR=NN)
        real*8::WORK(0:LWORK-1),AA2(0:LDA-1,0:NN-1)
        real*8::VL(0:NN-1,0:NN-1),VR(0:NN-1,0:NN-1)
        real*8::WR(0:NN-1),WI(0:NN-1)
        real*8::ev(NN),c2,x11,const,sigmay(0:3,0:3)
        real :: con
        sigmay(0,:)=(/0.0,0.0,0.0,-1.0/)
        sigmay(1,:)=(/0.0,0.0,1.0,0.0/)
        sigmay(2,:)=(/0.0,1.0,0.0,0.0/)
        sigmay(3,:)=(/-1.0,0.0,0.0,0.0/)
        site(0)=i0
        site(1)=j0
        call PTRVR(s,2,site,vin,rdm)
        write(3,*) rdm
        MM1=matmul(rdm,sigmay)
        MM2=matmul(rdm,sigmay)
        MM3=matmul(MM1,MM2)
        call dgeev('N', 'N', NN, MM3, LDA, WR, &
            WI, VL, LDVL, VR, LDVR, WORK, LWORK, INFO)
        do i=1,NN,1
            if (WR(i-1) .lt. 0.0000000000001) then
                WR(i-1)=0.0d0
            end if
                ev(i)=WR(i-1)
        end do
        do i=2,4
```

```
277             if(ev(i)>=ev(1))then
278                 evs=ev(i)
279                 ev(i)=ev(1)
280                 ev(1)=evs
281             end if
282         end do
283         do i=3,4
284             if(ev(i)>=ev(2))then
285                 evs=ev(i)
286                 ev(i)=ev(2)
287                 ev(2)=evs
288             end if
289         end do
290         do i=4,4
291             if(ev(i)>=ev(3))then
292                 evs=ev(i)
293                 ev(i)=ev(3)
294                 ev(3)=evs
295             end if
296         end do
297         c2=0
298         con=0
299         c2=dsqrt(ev(1))-dsqrt(ev(2))-dsqrt(ev(3))-dsqrt(ev(4))
300         ! print*, c2
301         if(c2>0)then
302             con=c2
303         else
304            con=0
305         end if
306     end subroutine
307     subroutine PTRVR(s, s1, site, vin, rdm)
308         implicit none
309         integer :: s, s1, s2
310         real*8 :: trace
311         real*8, dimension(0:2**s-1) :: vin
312         real*8, dimension(0:2**s1-1, 0:2**s1-1) :: rdm
313         integer :: ii, i1, a, i0, ia, j1, oo, k1, x1, y1, j, t1, t2,
    z1
```

```
314         integer, dimension(0:s1-1) :: site, site2
315         integer, dimension(0:s1-1) :: bin
316         integer, dimension(0:(2**(s-s1))*(2**s1)-1) :: ind
317         s2 = s - s1
318         x1 = 0
319         y1 = 0
320         ind = 0
321             do j1 = 2**s1-1, 0, -1
322                 do ii = 0, 2**s-1
323                     a = ii
324                     do i1 = 0, s1-1
325                         i0 = site(i1)
326                         call DTOBONEBIT(a, ia, i0, s)
327                         site2(i1) = ia
328                     end do
329                     call DTOB(j1, bin, s1)
330                     oo = 1
331                     do k1 = 0, s1-1
332                         oo = oo * (bin(k1) - site2(k1))
333                     end do
334                     if (abs(oo) == 1) then
335                         ind(x1) = ii
336                         x1 = x1 + 1
337                     end if
338                 end do
339             end do
340
341             rdm = 0.0d0
342             do t1 = 0, 2**s1-1
343                 do t2 = 0, 2**s1-1
344                     do z1 = 0, 2**s2-1
345                         rdm(t1, t2) = rdm(t1, t2) + vin(ind((2**s2)
    * t1 + z1)) * vin(ind((2**s2) * t2 + z1))
346                     end do
347                 end do
348             end do
349         end subroutine
350
```

```
351         subroutine DTOBONEBIT(m, ia, i0, s)
352             implicit none
353             integer*4 :: s
354             integer :: tt(0:s-1)
355             integer :: m, ia, i0
356             call DTOB(m, tt, s)
357             ia = tt(i0)
358         end subroutine
359 
360         subroutine DTOB(m, tt, s)
361             implicit none
362             integer :: s
363             integer, dimension(0:s-1) :: tt
364             integer :: m, k, a2
365             tt = 0
366             a2 = m
367             do k = 0, s-1
368                 tt(s-k-1) = mod(a2, 2)
369                 a2 = a2 / 2
370                 if (a2 == 0) exit
371             end do
372         end subroutine
373 end program Isinggen_with_bernoulli_periodic_chain_concurrence
```

We can also discuss the nature of concurrences for the periodic chain system. We have considered a three-qubit periodic chain system to discuss the variation of concurrences for ${}^3C_2 = 3$ different pairs of qubits viz. (1,2), (1,3) & (2,3) (We can also use this program to study higher spin systems). Here, the nature of variation is the same as that observed in the case of the open chain system. In Fig.[7.7], the variations are shown.

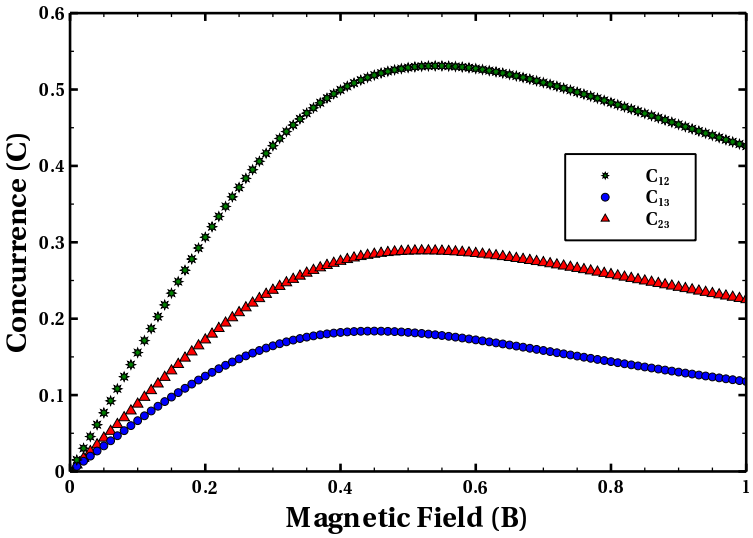


Figure 7.7: Variation of Concurrences with magnetic field for three-qubit periodic chain system.

In this case, we can also see that these variations are very similar to the variations obtained when we considered that the interactions between the nearest neighboring qubits of the spin chain system vary like inverse Fibonacci numbers in Chapter 4. We can observe this from the Fig.[7.8].

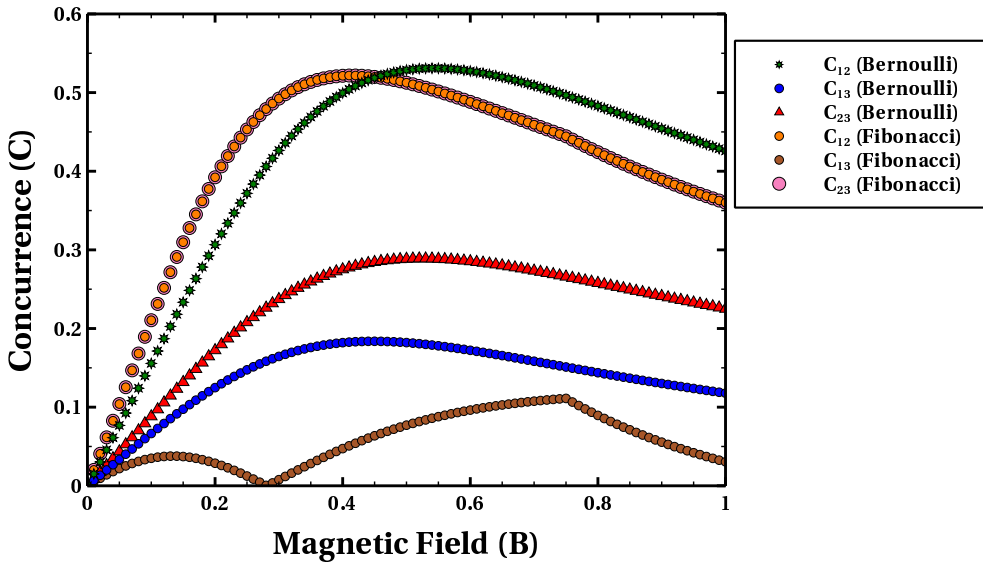


Figure 7.8: Variation of Concurrences with magnetic field for three-qubit periodic chain system (For both Inverse Fibonacci and Bernoulli type interaction).

So we have observed that the relation between the Bernoulli and the Fibonacci

series is reflected in the concurrences for the qubit pairs of the three-qubit open chain, as well as the periodic chain systems, which tends the concurrences to be varied in a similar manner. We are going to study this thing in more detail in the next chapter.

7.3.2 Study of von Neumann Entropy

In this section, we will see how the von Neumann entropy corresponding to the system varies with the external magnetic field. It is better to mention that we are interested in studying the 2-party entanglement. For that purpose, we have divided the total system into two blocks- one block corresponding to 2 qubits and another block corresponding to $(n-2)$ qubits (where 'n' is the total number of qubits comprising the system). Now, we are going to study the entanglement between these two blocks by calculating the von Neumann entropy of the system as per the definition provided in Eq.[4.5]. In the subsequent sections, we will show the variations for the two different spin-chain configurations.

7.3.2.1 Case(1) Open Chain System

We have constructed the following Fortran code to study the von Neumann entropy of the system in the open chain condition.

```
program Isinggen_with_Bernoulli_open_chain_entropy
!J is varied like the Bernoulli series, and B is also varied
!Concurrence calculation open chain
implicit none
integer :: n, i, num, a, x, y, k, t, c, differing_bits
real*8 :: B
real*8 :: be1
integer :: m
integer, parameter:: d = 2
character(len=:), allocatable :: state(:)
integer, allocatable :: decimal(:)
integer, allocatable :: psi(:)
double precision, allocatable :: p(:)
```

```
14 double precision, allocatable :: vec1(:), res1(:), vit(:)
15 integer :: j
16 real*8, allocatable :: S1(:,:), S2(:,:), H(:,:), H1(:,:), S3(:,:),
       rho(:,:)
17 integer, allocatable:: site(:)
18
19 ! Parameters for LAPACK DSYEV
20 integer:: lda, info, lwork
21 real*8, allocatable :: w(:), work(:)
22 character*1 :: JOBZ, UPLO
23
24 ! Input values
25 print*, "Enter the number of bits you want:"
26 read*, n
27
28 ! Calculate number of states
29 num = 2**n
30 m = n
31 ! Allocate arrays after determining the value of num
32 allocate(character(len=n) :: state(num))
33 allocate(integer :: decimal(num))
34 allocate(psi(n))
35 allocate(S1(num, num))
36 allocate(S2(num, num))
37 allocate(S3(num, num))
38 allocate(H(num, num))
39 allocate(H1(num, num))
40 allocate(p(0:m))
41 allocate(rho(num,num))
42 allocate(site(0:d-1))
43 allocate(vec1(num))
44 allocate(vit(0:num-1))
45
46 ! LAPACK parameters
47 lda = num
48 lwork = 3 * num - 1
49 allocate(real(8) :: w(lda), work(lwork))
50
```

```
! Printing the possible states
!print*, "The states are:"
do i = 0, num - 1
    call integer_binary(i, state(i + 1), n)  ! Adjusting index for
   1-based
    !print *, state(i + 1)
end do

! Convert binary states to decimal
do i = 1, num
    decimal(i) = btod(state(i), n)
end do

call Bernoulli_number(m, p)
write(9,*) p
! Perform calculations based on the decimal values for S1
do i = 1, num
    do j = 1, num
        call single_digit(state(i), psi)
        if (decimal(i) /= decimal(j)) then
            S1(i, j) = 0.0
        else
            S1(i, j) = 0.0
            do k = 2, n
                S1(i, j) = S1(i, j) + p(k-2)*(1.0 - 2.0 * psi(k-1))
   * (1.0 - 2.0 * psi(k))
            end do
        end if
    end do
end do

! Construct the S3 matrix (for sigma_x )
S3 = 0.0
do i = 1, num
    do j = 1, num
    ! Initialize differing_bits counter
        differing_bits = 0
        do k = 1, n
```

```
87              ! Compare the k-th bit of the i-th and j-th states
88              if (btest(decimal(i), k-1) .neqv. btest(decimal(j), k-1)
    ) then
89                  differing_bits = differing_bits + 1
90              end if
91          end do
92      ! If exactly one bit differs, set S2(i, j) = 1.0
93          if (differing_bits == 1) then
94              S3(i, j) = 1.0
95          else
96              S3(i, j) = 0.0
97          end if
98      end do
99  end do
100 
101 !Total Hamiltonian
102 !write(722,*) "H Array:"
103 do y = 0, 100, 1
104     B = y * 0.01
105     H = S1 + B*S3
106     !write(722,*)   H
107     ! Eigenvalue and eigenvector determination using DSYEV
108     JOBZ = 'V'
109     UPLO = 'U'
110     call DSYEV(JOBZ, UPLO, num, H, lda, w, work, lwork, info)
111     if (info == 0) then
112         do a = 1, lda
113            write(1225,*) B, w(a) !plot for magfield vs eigenvalue
114         end do
115     else
116         write(1225,*) 'Error in DSYEV, INFO=', info
117     end if
118     vec1 = H(:, 1)
119     do i = 1, num
120         vit(i-1) = vec1(i)
121     end do
122     ! Von Neumann Entropy
123     do i = 0, n-2
```

```
124         do j = i+1, n-1, 1
125             site(0) = i
126             site(1) = j
127             call BERPVR(n,2,site,vit,be1)
128             write(82,*) be1
129         end do
130     end do
131 end do
132 
133 deallocate(state)
134 deallocate(decimal)
135 deallocate(psi)
136 deallocate(S1)
137 deallocate(S2)
138 deallocate(S3)
139 deallocate(H)
140 deallocate(H1)
141 deallocate(p)
142 deallocate(rho)
143 deallocate(vit)
144 deallocate(vec1)
145 
146 contains
147     function btod(state, n) result(decimal)
148         implicit none
149         integer, intent(in) :: n
150         character(len=n) :: state
151         integer :: decimal, i
152         decimal = 0
153         do i = 1, n
154             if (state(i:i) == '1') then
155                 decimal = decimal + 2**(n-i)
156             end if
157         end do
158     end function btod
159 
160     subroutine integer_binary(num, binary_string, n)
161         implicit none
```

```
        integer :: num
        integer, intent(in) :: n
        integer :: i, temp_num
        character(len=n), intent(out) :: binary_string
        temp_num = num
        binary_string = repeat('0', n)
        do i = 0, n - 1
            if (mod(temp_num, 2) == 1) then
                binary_string(n-i:n-i) = '1'
            else
                binary_string(n-i:n-i) = '0'
            end if
            temp_num = temp_num / 2
        end do
    end subroutine integer_binary

    subroutine single_digit(A, psi)
        implicit none
        character(len=*), intent(in) :: A
        integer, intent(out) :: psi(:)
        integer:: i
        do i = 1, len(A)
            if (A(i:i) == '0') then
                psi(i) = 0
            else if (A(i:i) == '1') then
                psi(i) = 1
            else
                print *, "Invalid input in single_digit subroutine"
            end if
        end do
    end subroutine single_digit

    subroutine Bernoulli_number(n, B)
    implicit none
    double precision :: B(0:n)
    integer :: n, i, j, k
    double precision:: sum

```

```
200     if (n <= 0) then
201         print*, "Number of terms must be greater than 0."
202         stop
203     end if
204 
205     ! Initialization of the first term for the series
206     B(0) = 1.0d0  ! B(0) = 1
207 
208     ! Calculate the Bernoulli numbers
209     do j = 1, n
210         sum = 0.0d0
211         do k = 0, j - 1
212             sum = sum + (binomial_coefficient(j + 1, k)  * B(k))
213         end do
214         B(j) = -sum / (j + 1.0d0)
215     end do
216 
217     ! Print the Bernoulli series
218     !print*, "Bernoulli series:"
219     do i = 0, n
220         write(92,*) i, B(i)
221     end do
222 
223     end subroutine Bernoulli_number
224 
225     ! Function to calculate binomial coefficient
226     function binomial_coefficient(n, k) result(coefficient)
227         implicit none
228         integer, intent(in) :: n, k
229         double precision :: coefficient
230         integer :: i
231         coefficient = 1.0d0
232         if (k > 0 .and. k < n) then
233             do i = 1, k
234                 coefficient = coefficient * (n - i + 1) / i
235             end do
236         else if (k == 0 .or. k == n) then
237             coefficient = 1.0d0
```

```
238         else
239             coefficient = 0.0d0
240         end if
241     end function binomial_coefficient
242 
243     subroutine BERPVR(s,s1,site,vin,be)
244         implicit none
245         integer::s, s1,site(0:s1-1)
246         real*8::be,betemp,vin(0:2**s-1),rdm(0:2**s1-1,0:2**s1-1)
247         integer::i,INFO
248         real*8::WORK(0:3*(2**s1)-1),W(0:2**s1-1)
249         call PTRVR(s,s1,site,vin,rdm)
250         call DSYEV('N','U', 2**s1, rdm, 2**s1, &
251              W, WORK,3*(2**s1)-1,INFO)
252         be=0
253         betemp=0
254         do i=0,2**s1-1,1
255         if (W(i)>0.0000000000001) then
256         betemp=-(dlog(abs(W(i)))*abs(W(i)))/dlog(2.0d0)
257         be=be+betemp
258         end if
259         end do
260     end subroutine
261 
262     subroutine PTRVR(s, s1, site, vin, rdm)
263         implicit none
264         integer :: s, s1, s2
265         real*8 :: trace
266         real*8, dimension(0:2**s-1) :: vin
267         real*8, dimension(0:2**s1-1, 0:2**s1-1) :: rdm
268         integer :: ii, i1, a, i0, ia, j1, oo, k1, x1, y1, j, t1, t2,
    z1
269         integer, dimension(0:s1-1) :: site, site2
270         integer, dimension(0:s1-1) :: bin
271         integer, dimension(0:(2**(s-s1))*(2**s1)-1) :: ind
272         s2 = s - s1
273         x1 = 0
274         y1 = 0
```

```
275         ind = 0
276             do j1 = 2**s1-1, 0, -1
277                 do ii = 0, 2**s-1
278                     a = ii
279                     do i1 = 0, s1-1
280                         i0 = site(i1)
281                         call DTOBONEBIT(a, ia, i0, s)
282                         site2(i1) = ia
283                     end do
284                     call DTOB(j1, bin, s1)
285                     oo = 1
286                     do k1 = 0, s1-1
287                         oo = oo * (bin(k1) - site2(k1))
288                     end do
289                     if (abs(oo) == 1) then
290                         ind(x1) = ii
291                         x1 = x1 + 1
292                     end if
293                 end do
294             end do
295 
296             rdm = 0.0d0
297             do t1 = 0, 2**s1-1
298                 do t2 = 0, 2**s1-1
299                     do z1 = 0, 2**s2-1
300                         rdm(t1, t2) = rdm(t1, t2) + vin(ind((2**s2)
    * t1 + z1)) * vin(ind((2**s2) * t2 + z1))
301                     end do
302                 end do
303             end do
304         end subroutine
305 
306         subroutine DTOBONEBIT(m, ia, i0, s)
307             implicit none
308             integer*4 :: s
309             integer :: tt(0:s-1)
310             integer :: m, ia, i0
311             call DTOB(m, tt, s)
```

```
312             ia = tt(i0)
313         end subroutine
314
315         subroutine DTOB(m, tt, s)
316             implicit none
317             integer :: s
318             integer, dimension(0:s-1) :: tt
319             integer :: m, k, a2
320             tt = 0
321             a2 = m
322             do k = 0, s-1
323                 tt(s-k-1) = mod(a2, 2)
324                 a2 = a2 / 2
325                 if (a2 == 0) exit
326             end do
327         end subroutine
328 end program Isinggen_with_bernoulli_open_chain_entropy
```

Implementing the above code, we have plotted the variations of the von Neumann entropy in Fig.[7.9] by considering the first block consisting of (1,2),(1,3) & (2,3) qubit pairs and the other block consisting of the rest of the qubits, respectively, for the three-qubit open chain system. We can also use this program to study higher spin systems.

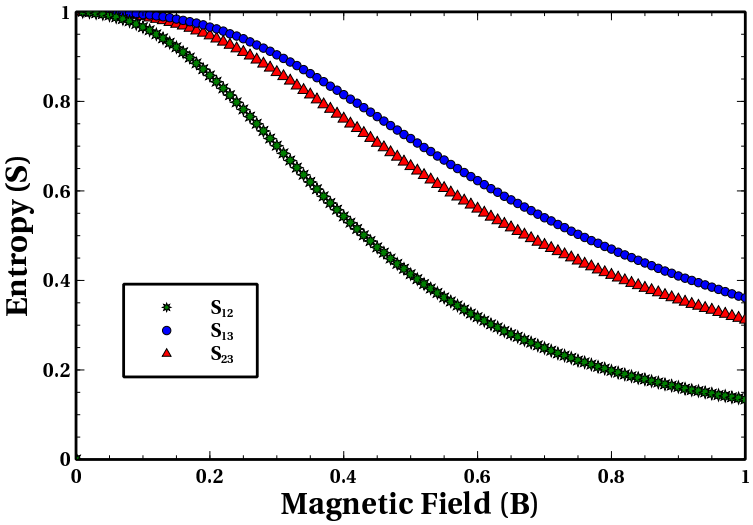


Figure 7.9: Variation of von Neumann entropy with the magnetic field for a three-qubit open chain system.

So we can see from the Fig.[7.9] that the von Neumann entropy corresponding to the block consisting (1,2),(1,3) & (2,3) decreases with the increase in the magnetic field which indicates that the entanglement of these blocks with the other block containing the remaining qubit (in this case 3^{rd}, 2^{nd} and & 1^{st} qubit respectively) decreases with the increase in the applied magnetic field. Now, the relation between Bernoulli and Fibonacci numbers (which has already been discussed in Chapter 6) suggests that the entanglement in these two chains is possibly related. If we carefully observe these variations and recall the variations of the same when we considered the Inverse-Fibonacci Ising-like spin system discussed in Chapter 4. We have plotted the variations of von Neumann entropy for these two spin chain systems simultaneously in Fig.[7.10].

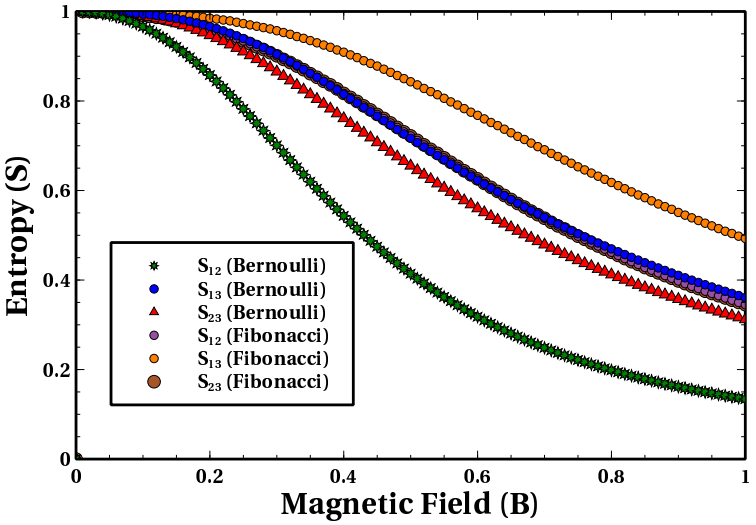


Figure 7.10: Variation of von Neumann entropy with magnetic field for a three-qubit open chain system (For both Inverse Fibonacci and Bernoulli type interaction).

Let us now discuss the same for the periodic chain system.

7.3.2.2 Case(2) Periodic Chain System

For the periodic chain system, we have to consider the periodic or wrap-up condition, which is $\sigma_{N+1} \equiv \sigma_1$. By imposing this condition, we can construct the following Fortran code to discuss the variations of von Neumann entropy for the system.

```
program Isinggen_with_Bernoulli_periodic_chain_entropy
!J is varied like the Bernoulli series, and B is also varied
!Concurrence calculation open chain
implicit none
integer :: n, i, num, a, x, y, k, t, c, differing_bits
real*8 :: B
real*8 :: be1
integer :: m
integer, parameter:: d = 2
character(len=:), allocatable :: state(:)
integer, allocatable :: decimal(:)
integer, allocatable :: psi(:)
double precision, allocatable :: p(:)
double precision, allocatable :: vec1(:), res1(:), vit(:)
integer :: j
real*8, allocatable :: S1(:,:), S2(:,:), H(:,:), H1(:,:), S3(:,:),
   rho(:,:)
integer, allocatable:: site(:)

! Parameters for LAPACK DSYEV
integer:: lda, info, lwork
real*8, allocatable :: w(:), work(:)
character*1 :: JOBZ, UPLO

! Input values
print*, "Enter the number of bits you want:"
read*, n

! Calculate number of states
num = 2**n
m = n
! Allocate arrays after determining the value of num
allocate(character(len=n) :: state(num))
allocate(integer :: decimal(num))
allocate(psi(n))
allocate(S1(num, num))
allocate(S2(num, num))
```

```
allocate(S3(num, num))
allocate(H(num, num))
allocate(H1(num, num))
allocate(p(0:m))
allocate(rho(num,num))
allocate(site(0:d-1))
allocate(vec1(num))
allocate(vit(0:num-1))

! LAPACK parameters
lda = num
lwork = 3 * num - 1
allocate(real(8) :: w(lda), work(lwork))

! Printing the possible states
!print*, "The states are:"
do i = 0, num - 1
    call integer_binary(i, state(i + 1), n)  ! Adjusting index for
   1-based
    !print *, state(i + 1)
end do

! Convert binary states to decimal
do i = 1, num
    decimal(i) = btod(state(i), n)
end do

call Bernoulli_number(m, p)
write(9,*) p
! Perform calculations based on the decimal values for S1
do i = 1, num
    do j = 1, num
        call single_digit(state(i), psi)
        if (decimal(i) /= decimal(j)) then
            S1(i, j) = 0.0
        else
        S1(i, j) = 0.0
        do k = 2, n
```

```
74              S1(i, j) = S1(i, j) + p(k-2)*(1.0 - 2.0 * psi(k-1)) *
    (1.0 - 2.0 * psi(k))
75          end do
76          S1(i, j) = S1(i, j) + p(n-1)*(1.0 - 2.0 * psi(n)) * (1.0 -
    2.0 * psi(1))
77          end if
78      end do
79  end do
80
81  ! Construct the S3 matrix (for sigma_x )
82  S3 = 0.0
83  do i = 1, num
84      do j = 1, num
85      ! Initialize differing_bits counter
86      differing_bits = 0
87          do k = 1, n
88              ! Compare the k-th bit of the i-th and j-th states
89              if (btest(decimal(i), k-1) .neqv. btest(decimal(j), k-1)
    ) then
90                  differing_bits = differing_bits + 1
91              end if
92          end do
93          if (differing_bits == 1) then
94              S3(i, j) = 1.0
95          else
96              S3(i, j) = 0.0
97          end if
98      end do
99  end do
100
101 !Total Hamiltonian
102 !write(722,*) "H Array:"
103 do y = 0, 100, 1
104     B = y * 0.01
105     H = S1 + B*S3
106     !write(722,*)   H
107     ! Eigenvalue and eigenvector determination using DSYEV
108     JOBZ = 'V'
```

```
109     UPLO = 'U'
110     call DSYEV(JOBZ, UPLO, num, H, lda, w, work, lwork, info)
111     if (info == 0) then
112         do a = 1, lda
113            write(1225,*) B, w(a) !plot for magfield vs eigenvalue
114         end do
115     else
116         write(1225,*) 'Error in DSYEV, INFO=', info
117     end if
118     vec1 = H(:, 1)
119     do i = 1, num
120         vit(i-1) = vec1(i)
121     end do
122     ! Von Neumann Entropy
123     do i = 0, n-2
124         do j = i+1, n-1, 1
125             site(0) = i
126             site(1) = j
127             call BERPVR(n,2,site,vit,be1)
128             write(83,*) be1
129         end do
130     end do
131 end do
132
133 deallocate(state)
134 deallocate(decimal)
135 deallocate(psi)
136 deallocate(S1)
137 deallocate(S2)
138 deallocate(S3)
139 deallocate(H)
140 deallocate(H1)
141 deallocate(p)
142 deallocate(rho)
143 deallocate(vit)
144 deallocate(vec1)
145
146 contains
```

```
147     function btod(state, n) result(decimal)
148         implicit none
149         integer, intent(in) :: n
150         character(len=n) :: state
151         integer :: decimal, i
152         decimal = 0
153         do i = 1, n
154             if (state(i:i) == '1') then
155                 decimal = decimal + 2**(n-i)
156             end if
157         end do
158     end function btod
159 
160     subroutine integer_binary(num, binary_string, n)
161         implicit none
162         integer :: num
163         integer, intent(in) :: n
164         integer :: i, temp_num
165         character(len=n), intent(out) :: binary_string
166         temp_num = num
167         binary_string = repeat('0', n)
168         do i = 0, n - 1
169             if (mod(temp_num, 2) == 1) then
170                 binary_string(n-i:n-i) = '1'
171             else
172                 binary_string(n-i:n-i) = '0'
173             end if
174             temp_num = temp_num / 2
175         end do
176     end subroutine integer_binary
177 
178     subroutine single_digit(A, psi)
179         implicit none
180         character(len=*), intent(in) :: A
181         integer, intent(out) :: psi(:)
182         integer:: i
183         do i = 1, len(A)
184             if (A(i:i) == '0') then
```

```
                psi(i) = 0
            else if (A(i:i) == '1') then
                psi(i) = 1
            else
                print *, "Invalid input in single_digit subroutine"
            end if
        end do
    end subroutine single_digit

    subroutine Bernoulli_number(n, B)
    implicit none
    double precision :: B(0:n)
    integer :: n, i, j, k
    double precision:: sum

    if (n <= 0) then
        print*, "Number of terms must be greater than 0."
        stop
    end if

    ! Initialization of the first term for the series
    B(0) = 1.0d0  ! B(0) = 1

    ! Calculate the Bernoulli numbers
    do j = 1, n
        sum = 0.0d0
        do k = 0, j - 1
            sum = sum + (binomial_coefficient(j + 1, k)  * B(k))
        end do
        B(j) = -sum / (j + 1.0d0)
    end do

    ! Print the Bernoulli series
    !print*, "Bernoulli series:"
    do i = 0, n
        write(92,*) i, B(i)
    end do

```

```
223     end subroutine Bernoulli_number
224
225     ! Function to calculate binomial coefficient
226     function binomial_coefficient(n, k) result(coefficient)
227         implicit none
228         integer, intent(in) :: n, k
229         double precision :: coefficient
230         integer :: i
231         coefficient = 1.0d0
232         if (k > 0 .and. k < n) then
233             do i = 1, k
234                 coefficient = coefficient * (n - i + 1) / i
235             end do
236         else if (k == 0 .or. k == n) then
237             coefficient = 1.0d0
238         else
239             coefficient = 0.0d0
240         end if
241     end function binomial_coefficient
242
243     subroutine BERPVR(s,s1,site,vin,be)
244         implicit none
245         integer::s, s1,site(0:s1-1)
246         real*8::be,betemp,vin(0:2**s-1),rdm(0:2**s1-1,0:2**s1-1)
247         integer::i,INFO
248         real*8::WORK(0:3*(2**s1)-1),W(0:2**s1-1)
249         call PTRVR(s,s1,site,vin,rdm)
250         call DSYEV('N','U', 2**s1, rdm, 2**s1, &
251              W, WORK,3*(2**s1)-1,INFO)
252         be=0
253         betemp=0
254         do i=0,2**s1-1,1
255         if (W(i)>0.0000000000001) then
256         betemp=-(dlog(abs(W(i)))*abs(W(i)))/dlog(2.0d0)
257         be=be+betemp
258         end if
259         end do
260     end subroutine
```

```
261
262     subroutine PTRVR(s, s1, site, vin, rdm)
263         implicit none
264         integer :: s, s1, s2
265         real*8 :: trace
266         real*8, dimension(0:2**s-1) :: vin
267         real*8, dimension(0:2**s1-1, 0:2**s1-1) :: rdm
268         integer :: ii, i1, a, i0, ia, j1, oo, k1, x1, y1, j, t1, t2,
    z1
269         integer, dimension(0:s1-1) :: site, site2
270         integer, dimension(0:s1-1) :: bin
271         integer, dimension(0:(2**(s-s1))*(2**s1)-1) :: ind
272         s2 = s - s1
273         x1 = 0
274         y1 = 0
275         ind = 0
276             do j1 = 2**s1-1, 0, -1
277                 do ii = 0, 2**s-1
278                     a = ii
279                     do i1 = 0, s1-1
280                         i0 = site(i1)
281                         call DTOBONEBIT(a, ia, i0, s)
282                         site2(i1) = ia
283                     end do
284                     call DTOB(j1, bin, s1)
285                     oo = 1
286                     do k1 = 0, s1-1
287                         oo = oo * (bin(k1) - site2(k1))
288                     end do
289                     if (abs(oo) == 1) then
290                         ind(x1) = ii
291                         x1 = x1 + 1
292                     end if
293                 end do
294             end do
295
296             rdm = 0.0d0
297             do t1 = 0, 2**s1-1
```

```
                do t2 = 0, 2**s1-1
                    do z1 = 0, 2**s2-1
                        rdm(t1, t2) = rdm(t1, t2) + vin(ind((2**s2)
    * t1 + z1)) * vin(ind((2**s2) * t2 + z1))
                    end do
                end do
            end do
        end subroutine

        subroutine DTOBONEBIT(m, ia, i0, s)
            implicit none
            integer*4 :: s
            integer :: tt(0:s-1)
            integer :: m, ia, i0
            call DTOB(m, tt, s)
            ia = tt(i0)
        end subroutine

        subroutine DTOB(m, tt, s)
            implicit none
            integer :: s
            integer, dimension(0:s-1) :: tt
            integer :: m, k, a2
            tt = 0
            a2 = m
            do k = 0, s-1
                tt(s-k-1) = mod(a2, 2)
                a2 = a2 / 2
                if (a2 == 0) exit
            end do
        end subroutine
end program Isinggen_with_bernoulli_periodic_chain_entropy
```

Implementing the above code, we have plotted the variations of the von Neumann entropy in Fig.[7.11] by considering the first block consisting of (1,2),(1,3) & (2,3) qubit pairs and the other block consisting of the rest of the qubits, respectively, for the three-qubit periodic chain system. We can also use this program to study higher

spin systems.

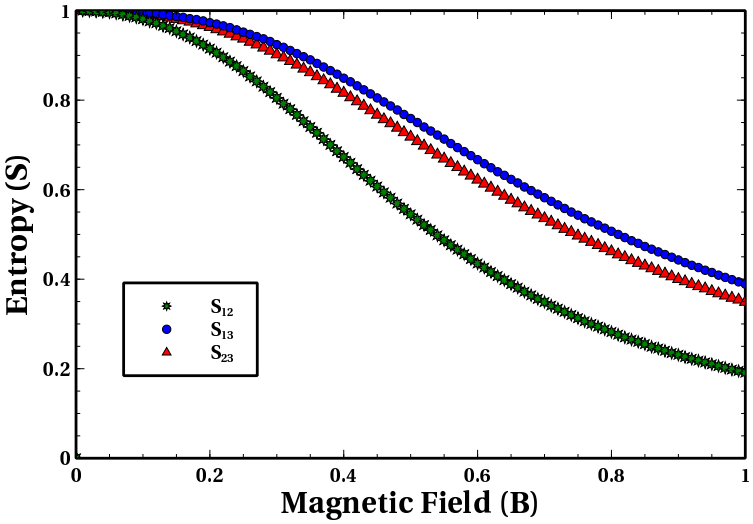


Figure 7.11: Variation of von Neumann entropy with the magnetic field for three-qubit periodic chain system.

We can see from the Fig.[7.9] that in the periodic chain condition also the von Neumann entropy corresponding to the block consisting (1,2),(1,3) & (2,3) decreases with the increase in the magnetic field which indicates that the entanglement of these blocks with the other block containing the remaining qubit (in this case 3^{rd}, 2^{nd} and & 1^{st} qubit respectively) decreases with the increase in the applied magnetic field. Now, the relation between Bernoulli and Fibonacci numbers (which has already been discussed in Chapter 6) suggests that the entanglement in these two chains is possibly related. In this case, we can observe the variations to be similar, recalling the variations of the same for the Inverse-Fibonacci Ising-like spin system discussed in Chapter 3. We have plotted the variations of von Neumann entropy for these two spin chain systems simultaneously in Fig.[7.12].

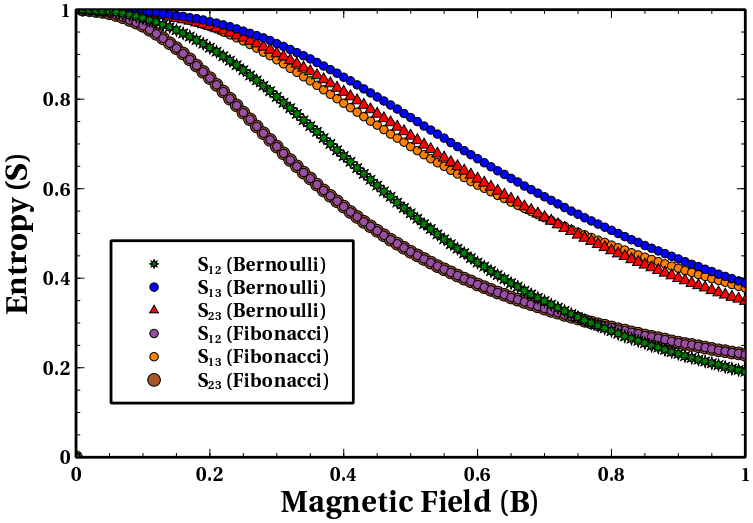
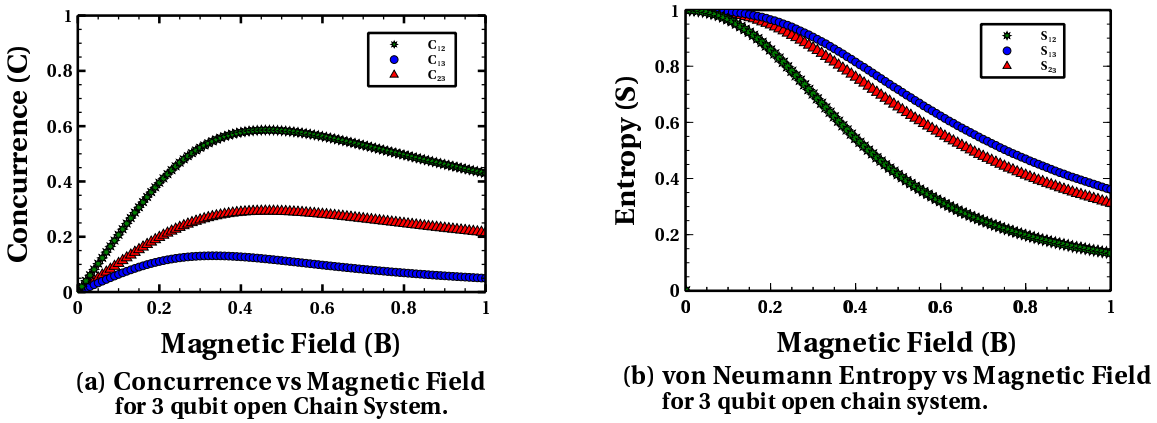


Figure 7.12: Variation of von Neumann entropy with magnetic field for three-qubit periodic chain system (For both Inverse Fibonacci and Bernoulli type interaction).

7.4 Monogamy of Entanglement (MOE)

We have discussed the variations of the concurrences for different qubit pairs, taking an example of a three-qubit system, both in the open chain and periodic chain conditions (We can also study higher spin systems). Again, we have also studied the variations of von Neumann entropy corresponding to the two blocks for a three-qubit open and periodic chain system. We have observed that the concurrences are increasing initially with the increase in the applied magnetic field implying the increase in entanglement between those qubit pairs & we also observed from the study of the von Neumann entropy that the entropy corresponding to the first block consisting of the qubit pairs decreases with the increase in the applied magnetic field indicating that the entanglement between these blocks with the other blocks containing the rest of the qubit decreases. This fact demonstrates the concept of monogamy of entanglement (MOE) as discussed in Chapter 4. We know that the MOE states that for multiparty entanglement, if there is a strong entanglement existing between two qubits, then the entanglement between the third qubit with any one of the previous two qubits will be very low and if the previous two qubits be in maximally entangled state (e.g. any one of the Bell states) then the entanglement of the third qubit with

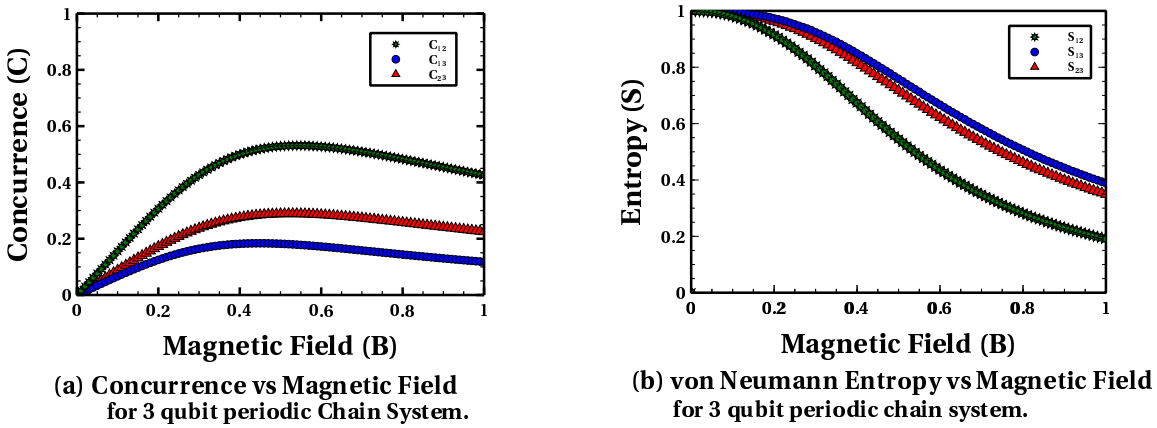
any one of them will be zero. We have shown the monogamy of entanglement for two cases of open chain and periodic chain systems, respectively, in Fig.[7.13] and Fig.[7.14].



(a) Concurrence vs Magnetic Field for 3 qubit open Chain System.

(b) von Neumann Entropy vs Magnetic Field for 3 qubit open chain system.

Figure 7.13: Monogamy of Entanglement(MOE) for three-qubit open chain system.



(a) Concurrence vs Magnetic Field for 3 qubit periodic Chain System.

(b) von Neumann Entropy vs Magnetic Field for 3 qubit periodic chain system.

Figure 7.14: Monogamy of Entanglement(MOE) for three-qubit periodic chain system.

We have observed that the monogamy of entanglement is also observed in the random Ising-like spin chain system when we have taken the Bernoulli numbers as the interaction strengths, which is remarkable.

7.5 Chapter Summary

We had started aiming to study the entanglement corresponding to the transverse Ising-like spin chain systems consisting of nearest neighbour interactions being taken as the terms of the Bernoulli series, making the model completely random. Our motivation was to study whether there is any effect on the entanglement between the qubit pairs for the two number-theoretic spin chain systems. Keeping this motivation in our mind, we constructed the random transverse Ising-like model involving Bernoulli numbers as the interaction strengths between consecutive qubit pairs and discussed the eigenvalue spectrum of this kind of system in both open as well as periodic chain configurations. We pointed out that the nature of variations of the concurrences and von Neumann entropies are very similar to that of our previous Ising-like system, where we had considered the interactions to be selected from the successive terms of the inverse Fibonacci series. We are also able to observe that the monogamy of entanglement is also followed when we consider this type of random spin chain. In the next chapter, we shall discuss more about the results obtained here.

Chapter 8

Analysis of The Numerical Results

8.1 Introduction

In the previous chapters, we had discussed the entanglement corresponding to the transverse Ising-like systems by considering the nearest neighbour interactions to be varied like inverse Fibonacci numbers (in Chapter 4) and like Bernoulli numbers (in Chapter 7) in both open as well as periodic chain configurations. We saw a strong resemblance in the results obtained for the two cases. In this chapter, we will discuss the results and try to find out the possible relations between the results obtained from the two systems.

8.2 Relation between Fibonacci and Bernoulli numbers

In Chapter 6, we discussed the relation between the Fibonacci and Bernoulli polynomials. One can ask, then, what is the relation between these two numbers, and also explore the results from the point of view of spin chain models under study.

We know that for the Fibonacci numbers

$$F_n = F_{n-1} + F_{n-2}. \tag{8.1}$$

Let the generating function for the Fibonacci numbers be expressed as,

$$\begin{aligned} G(x) &= \sum_{n=0}^{\infty} F_n\ x^n, \\ &= 0 + \sum_{n=1}^{\infty} F_n\ x^n, && [\text{since } F_0 = 0] \\ &= x + \sum_{n=2}^{\infty} F_n\ x^n, && [\text{since } F_1 = 1] \\ &= x + \sum_{n=2}^{\infty} (F_{n-1} + F_{n-2})\ x^n. \end{aligned} \tag{8.2}$$

Now,

$$\sum_{n=2}^{\infty} F_{n-1}\ x^n = x \sum_{n=2}^{\infty} F_{n-1}\ x^{n-1} = x \sum_{n=1}^{\infty} F_n\ x^n = xG(x). \tag{8.3}$$

Similarly,

$$\sum_{n=2}^{\infty} F_{n-2}\ x^n = x^2 \sum_{n=2}^{\infty} F_{n-2}\ x^{n-2} = x \sum_{n=0}^{\infty} F_n\ x^n = x^2\ G(x). \tag{8.4}$$

Therefore, Eq.[8.2] boils down to,

$$\begin{aligned} G(x) &= x + xG(x) + x^2\ G(x), \\ G(x) &= \frac{x}{1 - x - x^2}. \end{aligned} \tag{8.5}$$

Suppose we are considering an equation,

$$x^2 - x - 1 = 0 \tag{8.6}$$

$$\begin{aligned} x &= \frac{1 \pm \sqrt{(-1)^2 - 4 \cdot 1 \cdot (-1)}}{2 \cdot 1}, \\ &= \frac{1 \pm \sqrt{1 + 4}}{2}, \\ &= \frac{1 \pm \sqrt{5}}{2} \end{aligned} \tag{8.7}$$

Therefore, $\phi = \frac{1+\sqrt{5}}{2}$ and $\psi = \frac{1-\sqrt{5}}{2}$ are the two roots of the Eq.[8.6].

Again,

$$
\begin{aligned}
x &= x, \\
x^2 &= x + 1, && \text{[from equation } x^2 - x - 1 = 0\text{]} \\
x^3 &= x \cdot x^2 = x(x+1) = x^2 + x = 2x + 1, \\
x^4 &= x^2 \cdot x^2 = (x+1)(x+1) = x^2 + 2x + 1, \\
&= (x+1) + 2x + 1 = 3x + 2, \\
x^5 &= x^2 \cdot x^3 = (x+1)(2x+1), \\
&= 2x^2 + 2x + x + 1 = 2x^2 + 3x + 1, \\
&= 2(x+1) + 3x + 1 = 5x + 3, \\
&\vdots
\end{aligned}
$$

By close observation, we can write then

$$
\begin{aligned}
x^1 &= F_1\ x, \\
x^2 &= F_2\ x + F_1, && \text{[from equation } x^2 - x - 1 = 0\text{]} \\
x^3 &= x \cdot x^2 = x(x+1) = x^2 + x, \\
&= F_3\ x + F_2, \\
x^4 &= x^2 \cdot x^2 = (x+1)(x+1) = x^2 + 2x + 1, \\
&= (x+1) + 2x + 1, \\
&= F_4\ x + F_3, \\
x^5 &= x^2 \cdot x^3 = (x+1)(2x+1), \\
&= 2x^2 + 2x + x + 1 = 2x^2 + 3x + 1, \\
&= 2(x+1) + 3x + 1, \\
&= F_5\ x + F_4, \\
&\vdots
\end{aligned}
$$

Therefore, in general, we can write,

$$x^n = F_n\ x + F_{n-1}. \tag{8.8}$$

Now, $\phi = \frac{1+\sqrt{5}}{2}$ and $\psi = \frac{1-\sqrt{5}}{2}$ are the two roots of the Eq.[8.6] which implies that,

$$\phi^n = F_n\ \phi + F_{n-1}, \tag{8.9}$$

$$\psi^n = F_n\ \psi + F_{n-1}. \tag{8.10}$$

Substituting F_{n-1} from Eq.[8.10] to Eq.[8.9],

$$\phi^n = F_n\ \phi + \psi^n - F_n\ \psi. \tag{8.11}$$

Therefore, Eq.[8.11] boils down to,

$$\boxed{F_n = \frac{\phi^n - \psi^n}{\phi - \psi}}. \tag{8.12}$$

This is known as the closed form expression of Fibonacci numbers, in short also called *Binet's form.*

Let us consider,

$$e^{\phi t} = \sum_{n=0}^{\infty} \frac{(\phi t)^n}{n!}, \tag{8.13}$$

$$e^{\psi t} = \sum_{n=0}^{\infty} \frac{(\psi t)^n}{n!}. \tag{8.14}$$

Since we know from the Taylor series expansion of e^x, which is given by

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}. \tag{8.15}$$

Therefore, the equation 8.13 and 8.14 we can write,

$$\begin{aligned}
e^{\phi t} - e^{\psi t} &= \sum_{n=0}^{\infty} \frac{(\phi^n - \psi^n)t^n}{n!}, \\
\frac{e^{\phi t} - e^{\psi t}}{\phi - \psi} &= \sum_{n=0}^{\infty} \frac{(\phi^n - \psi^n)}{(\phi - \psi)} \frac{t^n}{n!}, \\
\frac{e^{\phi t} - e^{\psi t}}{\phi - \psi} &= \sum_{n=0}^{\infty} F_n\ \frac{t^n}{n!} \qquad \text{(from Eq.[8.12]).}
\end{aligned} \tag{8.16}$$

Now,

$$\begin{aligned}
\phi - \psi &= \left(\frac{1+\sqrt{5}}{2}\right) - \left(\frac{1-\sqrt{5}}{2}\right), \\
&= \frac{2\sqrt{5}}{2}, \\
&= \sqrt{5}
\end{aligned} \tag{8.17}$$

From Eq.[8.16], we can get,

$$\begin{aligned}
\frac{e^{\phi t}-e^{\psi t}}{\sqrt{5}} &= \sum_{n=0}^{\infty} F_n \frac{t^n}{n!},\\
e^{\phi t}-e^{\psi t} &= \sqrt{5}\sum_{n=0}^{\infty} F_n \frac{t^n}{n!},\\
e^{\phi t}(1-e^{(\psi-\phi)t}) &= \sqrt{5}\sum_{n=0}^{\infty} F_n \frac{t^n}{n!},\\
e^{\phi t}(1-e^{-\sqrt{5}t}) &= \sqrt{5}\sum_{n=0}^{\infty} F_n \frac{t^n}{n!},\\
te^{\phi t} &= \frac{-t\sqrt{5}}{e^{-t\sqrt{5}}-1}\sum_{n=0}^{\infty} F_n \frac{t^n}{n!}.
\end{aligned} \tag{8.18}$$

Again, from the generating function of the Bernoulli numbers given in Eq.[6.8], which we had already discussed in Chapter 6, we can write,

$$\frac{-t\sqrt{5}}{e^{-t\sqrt{5}}-1} = \sum_{k=0}^{\infty} \frac{B_k(-t\sqrt{5})^k}{k!}, \quad \left(\text{since} \quad \frac{x}{e^x-1} = \sum_{k=0}^{\infty} B_k \frac{x^k}{k!}\right). \tag{8.19}$$

Therefore Eq.[8.18] boils down to,

$$\begin{aligned}
te^{\phi t} &= \sum_{k=0}^{\infty} \frac{B_k(-t\sqrt{5})^k}{k!}\sum_{n=0}^{\infty} F_n \frac{t^n}{n!},\\
t\sum_{n=0}^{\infty}\frac{(\phi t)^n}{n!} &= \sum_{k=0}^{\infty} \frac{B_k(-t\sqrt{5})^k}{k!}\sum_{n=0}^{\infty} F_n \frac{t^n}{n!}, && \text{(from Eq.[8.13])}\\
&= \sum_{n=0}^{\infty}\sum_{k=0}^{n} \frac{B_k(-\sqrt{5})^k t^k}{k!}\frac{F_{n-k}t^{n-k}}{(n-k)!}, && \text{(using Eq.[6.12])}\\
\sum_{n=0}^{\infty}\frac{\phi^n t^n}{n!} &= \sum_{n=0}^{\infty}\sum_{k=0}^{n} \frac{B_k(-\sqrt{5})^k}{k!}\frac{F_{n-k}t^{n-1}}{(n-k)!}.
\end{aligned} \tag{8.20}$$

Comparing the coefficient of t^n on both sides, we can get,

$$\begin{aligned}
\frac{\phi^n}{n!} &= \sum_{k=0}^{n} \frac{B_k(-\sqrt{5})^k}{k!}\frac{F_{n-k+1}}{(n-k+1)!},\\
&= \sum_{k=0}^{n} \frac{B_k(-\sqrt{5})^k}{(n-k)!\;k!}\frac{F_{n-k+1}}{n-k+1},\\
\phi^n &= \sum_{k=0}^{n} \frac{B_k(-\sqrt{5})^k n!}{(n-k)!\;k!}\frac{F_{n-k+1}}{n-k+1},\\
&= \sum_{k=0}^{n} B_k(-\sqrt{5})^k \frac{n!}{(n-k)!\;k!}\frac{F_{n-k+1}}{n-k+1},\\
&= \sum_{k=0}^{n} B_k(-\sqrt{5})^k\; {}^nC_k \frac{F_{n-k+1}}{n-k+1}.
\end{aligned} \tag{8.21}$$

Therefore,

$$\sum_{k=0}^{n} {}^{n}C_k \; B_k(-\sqrt{5})^k \frac{F_{n-k+1}}{n-k+1} = \phi^n \quad . \tag{8.22}$$

Let us check this equation with the help of Mathematica by placing different values of n.

8.2.0.1 Case(1) For some low value of n (e.g. $n = 8$)

Mathematica

L.H.S.

$$In[1] := \text{Sum}\left[\frac{\left(-\sqrt{5}\right)^k B_k \binom{n}{k} F_{n-k+1}}{n-k+1}, \{k, 0, n, 1\}\right] \;/.\; n \to 8$$

$$Out[1] := \frac{1}{256}\left(\sqrt{5}+1\right)^8$$

$$In[2] := N\left[\frac{1}{256}\left(\sqrt{5}+1\right)^8\right]$$

$$Out[2] := 46.9787$$

R.H.S.

$$In[3] := \left(\frac{1}{2}\left(\sqrt{5}+1\right)\right)^n \;/.\; n \to 8$$

$$Out[3] := \frac{1}{256}\left(\sqrt{5}+1\right)^8$$

$$In[4] := N\left[\frac{1}{256}\left(\sqrt{5}+1\right)^8\right]$$

$$Out[4] := 46.9787$$

8.2.0.2 Case(2) For some medium value of n (e.g. $n = 30$)

Mathematica

L.H.S.

$$In[5] := \text{Sum}\left[\frac{\left(-\sqrt{5}\right)^k B_k\binom{n}{k} F_{n-k+1}}{n-k+1}, \{k, 0, n, 1\}\right] \;/.\; n \to 30$$

$$Out[5] := 416020\sqrt{5} + 930249$$

$$In[6] := N\left[416020\sqrt{5} + 930249\right]$$

$$Out[6] := 1.8605 * 10^6$$

R.H.S.

$$In[7] := \left(\frac{1}{2}\left(\sqrt{5}+1\right)\right)^n \;/.\; n \to 30$$

$$Out[7] := \frac{\left(\sqrt{5}+1\right)^{30}}{1073741824}$$

$$In[8] := N\left[\frac{\left(\sqrt{5}+1\right)^{30}}{1073741824}\right]$$

$$Out[8] := 1.8605 * 10^6$$

NOTE: We can set our precision order in Mathematica by the command **SetPrecision**[**expression**, **p**(**the decimal point up to which you want the precision**)]

L.H.S.

$$In[5] := \text{SetPrecision}\left[\text{Sum}\left[\frac{\left(-\sqrt{5}\right)^k B_k\binom{n}{k} F_{n-k+1}}{n-k+1}, \{k, 0, n, 1\}\right] \;/.\; n \to 30, 20\right]$$

$$Out[5] := 1.8604979999994625095 * 10^6$$

R.H.S.

$$In[7] := \text{SetPrecision}\left[\left(\frac{1}{2}\left(\sqrt{5}+1\right)\right)^n \;/.\; n \to 30, 20\right]$$

$$Out[7] := 1.8604979999999462510 * 10^6$$

8.2.0.3 Case(3) For some large value of n (e.g. $n = 70$)

Mathematica

L.H.S.

$$In[8] := \text{SetPrecision}\left[\text{Sum}\left[\frac{\left(-\sqrt{5}\right)^k B_k\binom{n}{k} F_{n-k+1}}{n-k+1}, \{k, 0, n, 1\}\right] \;/.\; n \to 70, 80\right]$$

$$Out[8] := 4.25730551631122999999999999997651096459$$
$$55957492266835789305442778434643467673813 * 10^{14}$$

R.H.S.

$$In[9] := \text{SetPrecision}\left[\left(\frac{1}{2}\left(\sqrt{5}+1\right)\right)^n \;/.\; n \to 70, 80\right]$$

$$Out[9] := 4.25730551631122999999999999997651096459$$
$$55957492266835789305442778434643467673813 * 10^{14}$$

$$In[10] := \text{SetPrecision}[Out[8] - Out[9], 40]$$

$$Out[10] :0$$

We can see that the relation Eq.[8.22] is valid for all the values of n from a very small value to a large value.

Let us check the Eq.[8.22] with a Fortran code as given below.

```
program verification
implicit none
integer :: n, i, j, p, m
real(16), allocatable :: B(:), fib(:)
real(16) :: LHS, RHS
! Input the value of n
print*, "Value of n:"
read*, n              !The upper limit of the sum
print*, "Value of m:"
read*, m    !The dimension of the Fibonacci array
print*, "Value of p:"
read*, p    !The dimension of the Bernoulli array

```

```
! Allocate the arrays
allocate(B(0:m))
allocate(fib(0:p))

! Calculate the Bernoulli and Fibonacci numbers
call Bernoulli_number(m, B)
call FibonacciSeries(p, fib)

! Compute the LHS
LHS = 0.0_16
do i = 0, n
    LHS = LHS + binomial_coefficient(n, i) * ((-sqrt(5.0_16))**i) *
   (fib(n-i+1) / real(n-i+1, 16)) * B(i)
end do
write(100,*) "LHS = ", LHS
write(100,*)

! Compute the RHS
RHS = (0.5_16 * (1+sqrt(5.0_16)))**n
write(100,*) "RHS = ", RHS

! Deallocate arrays
deallocate(B)
deallocate(fib)

contains
! Function to compute binomial coefficients
function binomial_coefficient(n, k) result(coefficient)
implicit none
integer, intent(in) :: n, k
real(16) :: coefficient
integer :: i

if (k < 0 .or. k > n) then
    coefficient = 0.0_16
else if (k == 0 .or. k == n) then
    coefficient = 1.0_16
else
```

```
    coefficient = 1.0_16
    do i = 1, k
        coefficient = coefficient * (n - i + 1) / real(i, 16)
    end do
end if
end function binomial_coefficient

! Subroutine to compute Fibonacci series
subroutine FibonacciSeries(m, fib)
implicit none
integer, intent(in) :: m
real(16), intent(out) :: fib(0:m)
integer :: i

if (m < 0) then
    print*, "Number of terms must be greater than 0."
    stop
end if

! Initialize the first two terms
fib(0) = 0.0_16
    if (m > 0) fib(1) = 1.0_16
    ! Generate the Fibonacci series
    do i = 2, m
        fib(i) = fib(i-1) + fib(i-2)
    end do
end subroutine FibonacciSeries

! Subroutine to compute Bernoulli numbers
subroutine Bernoulli_number(n, B)
implicit none
integer, intent(in) :: n
real(16), intent(out) :: B(0:n)
integer :: i, j, k
real(16) :: sum

if (n < 0) then
    print*, "Number of terms must be greater than 0."
```

```
    stop
end if

! Initialize the first term for the series
    B(0) = 1.0_16  ! B(0) = 1
! Calculate the Bernoulli numbers
    do j = 1, n
    sum = 0.0_16
        do k = 0, j - 1
            sum = sum + binomial_coefficient(j + 1, k) * B(k)
        end do
        B(j) = -sum / real(j + 1, 16)
    end do
! Print the Bernoulli series
    do i = 0, n
        write(92,*) i, B(i)
    end do
end subroutine Bernoulli_number
end program verification
```

Let us see the output for n=41

```
 LHS =    370248451.09925078349202469645400312 9

 RHS =    370248451.00000000270088908488100616 5
```

So we have verified the Eq.[8.22] for $n = 41$ in high double precission.

8.3 Numerical study to determine the relation between the results obtained from the two different transverse Ising-like Models involving Bernoulli and inverse Fibonacci type of interactions

The results obtained from Chapter 7 can now be analysed with the results obtained in Chapters 3 and 4 together to find any possible results from the two models involving two different successive interaction strengths. If we succeed in solving these random Ising-like models analytically, then our job would become easier. We know that the Jordan-Wigner transformation[29] is available to solve the transverse Ising-like systems involving constant interaction between the spins in that system. Our model is a random model where the interaction strengths are varied from site to site . Therefore, we can not apply the Jordan-Wigner transformation to our model to solve it analytically. Now, we have our numerical results for the two models. So, we can interpolate the results corresponding to the two models and get some interesting relations.

8.3.1 Relation between the Ground state Energies of the two systems

We have already seen that the nature of the ground state energy variations with the magnetic field for both cases is similar. So we can now try the technique of interpolation[30] between the two ground state energies to find some relations between them. For that, we can consider a three-qubit open chain system. The following table corresponds to the ground-state energies of the two Ising-like models involving Bernoulli and inverse Fibonacci type of interactions in open chain configuration for 100 iterations by varying the magnetic field (B) from 0 to 1 in steps of 0.01 units.

Ground state energy (Bernoulli interaction)	Ground state energy (inverse Fibonacci interaction)
-1.5	-2
-1.50018533119784	-2.00012600098678
-1.50074929094382	-2.0005080141928
-1.50170374234069	-2.00115206328214
-1.50306037961541	-2.00206417212228
-1.50483065701046	-2.00325035019587
-1.5070257167089	-2.00471657859436
-1.50965630946724	-2.00646879180242
-1.51273271819109	-2.00851286149138
-1.51626468316553	-2.01085458070518
-1.52026132270431	-2.013499643542
-1.52473106377366	-2.0164536291945
-1.52968155847763	-2.01972197358366
-1.53511963950851	-2.02330996289138
-1.54105125390959	-2.02722270816525
-1.54748139318325	-2.03146511230373
-1.55441410359053	-2.03604188416373
-1.56185239349532	-2.04095747856389
-1.56979823785117	-2.04621609390804
-1.57825260706464	-2.05182167952768
-1.58721538312106	-2.05777786191376
-1.59668547206261	-2.06408799493642
-1.6066607344718	-2.07075508355997
-1.61713805397224	-2.07778179395181

Ground state energy (Bernoulli interaction)	Ground state energy (inverse Fibonacci interaction)
-1.62811342815252	-2.08517047465571
-1.63958192343759	-2.09292308277717
-1.65153777387458	-2.10104120372122
-1.66397447971202	-2.10952606963904
-1.67688485982533	-2.11837854602089
-1.6902609809327	-2.12759903295143
-1.70409445798312	-2.13718762205292
-1.71837641489018	-2.14714402298593
-1.73309742021713	-2.1574674722571
-1.74824781064982	-2.16815691602352
-1.76381763485344	-2.1792109326932
-1.77979657137098	-2.19062763775422
-1.79617426647819	-2.20240489175848
-1.81294025586338	-2.21454021826185
-1.83008386077716	-2.22703070390788
-1.84759453496623	-2.23987323017657
-1.86546170863880	-2.25306434567701
-1.88367493099227	-2.26660035951761
-1.90222368613519	-2.28047719661204
-1.92109774875786	-2.29469065807268
-1.94028705166450	-2.30923632349814
-1.95978153213502	-2.32410943549901
-1.97957149391956	-2.33930517903264
-1.99964745524451	-2.35481857487631

Ground state energy (Bernoulli interaction)	Ground state energy (inverse Fibonacci interaction)
-2.01999997801897	-2.3706443556304
-2.04062003684583	-2.3867772618757
-2.06149885068464	-2.40321192591155
-2.08262769799449	-2.4199427385021
-2.10399829359331	-2.43696415996783
-2.12560254228772	-2.45427054165662
-2.14743260101578	-2.4718561902334
-2.16948100991412	-2.48971548989064
-2.19174016324551	-2.5078424890348
-2.21420309493778	-2.52623155228735
-2.23686295033143	-2.54487694865236
-2.25971311246649	-2.563772968643
-2.28274719705411	-2.58291393488395
-2.30595918588590	-2.60229432805725
-2.32934286655309	-2.62190833227927
-2.35289265656154	-2.64175053628053
-2.37660304469046	-2.66181547996
-2.40046872245508	-2.68209777564661
-2.42448457767111	-2.7025921138113
-2.44864568799702	-2.72329326807449
-2.47294745975259	-2.7441962247288
-2.49738504129152	-2.76529568688357
-2.52195418582613	-2.78658682518103
-2.54665066662709	-2.80806478317948

Ground state energy (Bernoulli interaction)	Ground state energy (inverse Fibonacci interaction)
-2.57147041537016	-2.82972480341934
-2.59640951618404	-2.85156222902501
-2.62146434952105	-2.87357263657275
-2.64663098846741	-2.89575131106459
-2.67190608931890	-2.91809403353168
-2.69728628957903	-2.94059655735354
-2.72276835181356	-2.96325473873014
-2.74834915868120	-2.98606453620745
-2.77402570813771	-3.0090220099655
-2.79979526268134	-3.03212345901126
-2.82565472995321	-3.05536486842986
-2.85160158006398	-3.07874273431325
-2.87763322761689	-3.10225351239726
-2.90374718107323	-3.12589375379354
-2.9299410389515	-3.14966010334719
-2.9562126429965	-3.17354944065673
-2.98255944785507	-3.19755830797221
-3.00897945728189	-3.22168376270578
-3.03547059500932	-3.24592280666854
-3.06203085802696	-3.2702725261534
-3.08865831362961	-3.29473008996474
-3.11535125588258	-3.31929289414468
-3.14210756611742	-3.34395797372993
-3.16892566496528	-3.36872287921031

Ground state energy (Bernoulli interaction)	Ground state energy (inverse Fibonacci interaction)
-3.19580387384627	-3.39358508952693
-3.22274057150326	-3.41854215618869
-3.24973419172889	-3.44359170131075
-3.2767832211902	-3.46873141567391
-3.30388635905125	-3.49395920743494

Table 8.1: Ground state energies for Bernoulli and inverse Fibonacci interactions

We have stored these data points in a text file (`interpolation_gs_varying_B.txt`) and written the following Python code to do the interpolation and obtain the polynomial fitted curve.

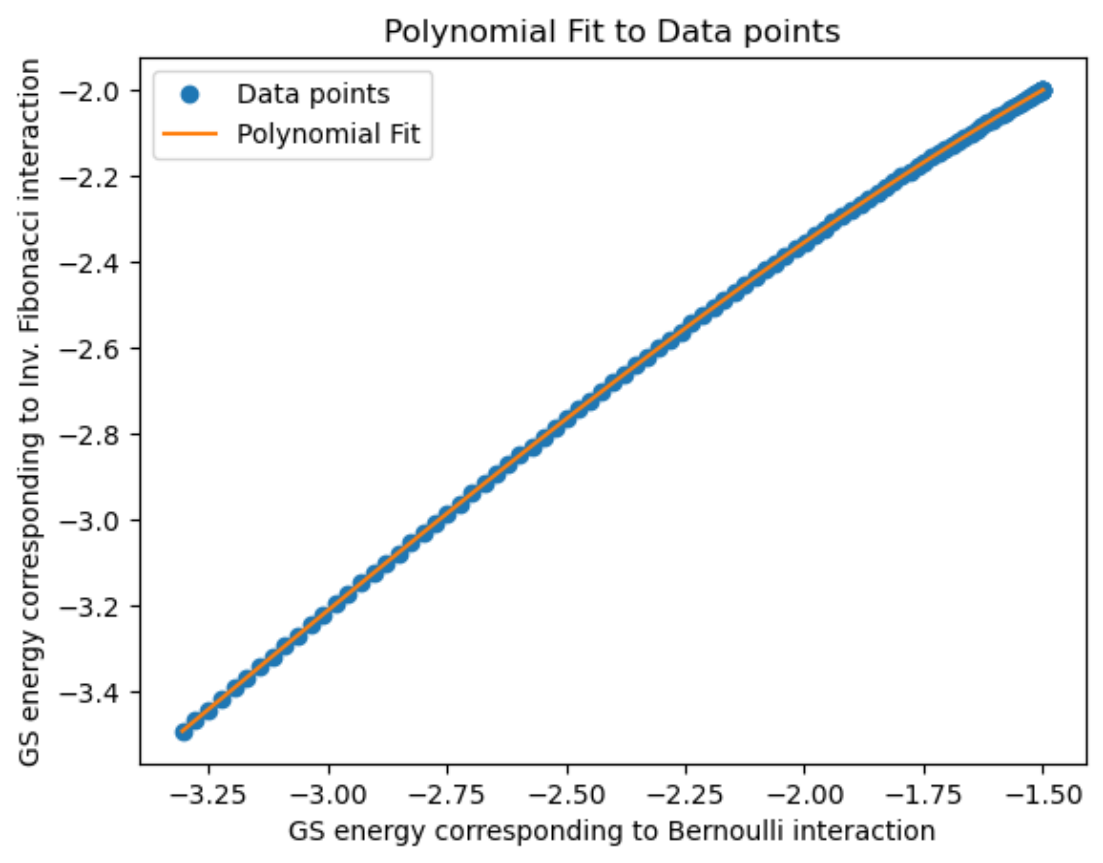
```
import numpy as np
import matplotlib.pyplot as plt
import numpy.polynomial.polynomial as poly
from scipy.interpolate import interp1d
# Load data
data = np.loadtxt("interpolation_gs_varying_B.txt")
x = data[:, 0]
y = data[:, 1]
coeffs = poly.polyfit(x, y, 4)
poly_eqn = np.poly1d(coeffs)
yfit = poly.polyval(x, coeffs)

# Plotting
plt.plot(x, y, 'o', label="Data points") #Data points plotting
plt.plot(x, yfit, '-', label="Fitted Curve") # Fitted polynomial
plt.xlabel("GS energy corresponding to Bernoulli interaction")
plt.ylabel("GS energy corresponding to Inv. Fibonacci interaction")
plt.legend()
plt.title("Polynomial Fit to Data points")
plt.show()
coeffs = np.polyfit(x, y, 4)
```

```
22
23 # Print polynomial coefficients
24 print("Polynomial Coefficients:")
25 print(coeffs)
26 poly_eqn = np.poly1d(coeffs)
27
28 # Print the polynomial equation
29 print("Polynomial Equation:")
30 print(poly_eqn)
```

Python outputs

Output 1:



Python outputs

Output 2:

Polynomial Coefficients:

$[-0.00232785 - 0.05111642 - 0.36444672 - 0.14405493 - 1.55742602]$

Output 3:

Polynomial Equation:

$-0.002328x^4 - 0.05112x^3 - 0.3644x^2 - 0.1441x - 1.557$

We have plotted these data points in Fig.[8.1], and by the process of curve fitting, we have obtained that the fitted function is given by,

$$y = ax^4 + bx^3 + cx^2 + dx + e \tag{8.23}$$

where $a = -0.002328$, $b = -0.05112$, $c = -0.3644$, $d = -0.1441$, $e = -1.557$, considering the X-data as the data obtained from the first column and Y-data as the data obtained from the second column of the table mentioned above.

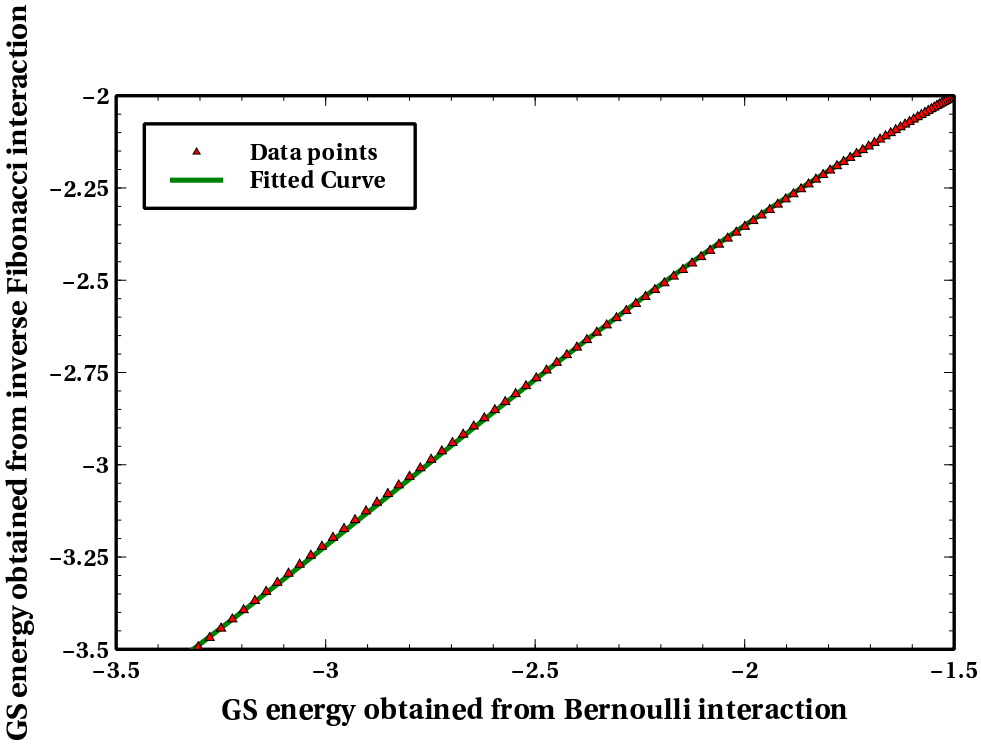


Figure 8.1: Interpolation of ground state energies obtained from Bernoulli and inverse Fibonacci interactions.

From Eq.[8.23] replacing y with the ground state energy corresponding to the inhomogeneous transverse one-dimensional Ising-like model with inverse Fibonacci-like interactions ($(E_{gs})_{inv.Fibo}$) and x with the ground state energy corresponding to the inhomogeneous transverse one-dimensional Ising-like model with Bernoulli number-like interactions ($(E_{gs})_{Bernoulli}$), we can write that,

$$(E_{gs})_{inv.Fibo} = a((E_{gs})_{Bernoulli})^4 + b((E_{gs})_{Bernoulli})^3 + c((E_{gs})_{Bernoulli})^2 + d((E_{gs})_{Bernoulli}) + e$$

where,$a = -0.002328$, $b = -0.05112$, $c = -0.3644$, $d = -0.1441$, $e = -1.557$ respectively.

8.3.2 Relation between the Concurrences

Let us see how the concurrences corresponding to the qubit pairs of the two systems of different sets of interaction energies are related. For that purpose, we have considered the three-qubit open chain system and calculated concurrences corresponding to the possible qubit pairs, like in this case these are C_{12}(concurrence corresponding to 1st and 2nd qubit), C_{23}(concurrence corresponding to 2nd and 3rd qubit) & C_{13}(concurrence corresponding to 1st and 3rd qubit).

C_{12}		C_{23}		C_{13}	
Bernoulli	Inv.Fibo.	Bernoulli	Inv. Fibo.	Bernoulli	Inv. Fibo.
0	0	0	0	0	0
0.02	0.01	0.01	0.01	0.006	0.0050
0.04	0.02	0.02	0.02	0.013	0.0099
0.06	0.03	0.03	0.03	0.019	0.0149
0.08	0.04	0.04	0.04	0.026	0.0199
0.10	0.05	0.05	0.05	0.033	0.0248
0.12	0.06	0.06	0.06	0.039	0.0298

C_{12}		C_{23}		C_{13}	
Bernoulli	Inv.Fibo.	Bernoulli	Inv. Fibo.	Bernoulli	Inv. Fibo.
0.14	0.07	0.07	0.07	0.045	0.0347
0.16	0.08	0.08	0.08	0.052	0.0395
0.18	0.09	0.09	0.09	0.058	0.0443
0.20	0.10	0.10	0.10	0.064	0.0490
0.22	0.11	0.11	0.11	0.069	0.0537
0.24	0.12	0.12	0.12	0.075	0.0584
0.26	0.13	0.13	0.13	0.080	0.0629
0.28	0.14	0.14	0.14	0.085	0.0674
0.30	0.15	0.15	0.15	0.090	0.0718
0.32	0.16	0.16	0.16	0.095	0.0761
0.34	0.17	0.17	0.17	0.099	0.0803
0.36	0.18	0.18	0.18	0.103	0.0844
0.37	0.19	0.18	0.19	0.107	0.0884
0.39	0.20	0.19	0.20	0.110	0.0922
0.41	0.22	0.20	0.21	0.114	0.0960
0.42	0.23	0.21	0.22	0.117	0.0996
0.44	0.24	0.22	0.23	0.119	0.1031
0.45	0.25	0.22	0.24	0.121	0.1065
0.46	0.26	0.23	0.25	0.123	0.1097
0.48	0.27	0.24	0.26	0.125	0.1127
0.49	0.27	0.24	0.27	0.127	0.1157
0.50	0.28	0.25	0.28	0.128	0.1184
0.51	0.29	0.25	0.29	0.129	0.1210
0.52	0.30	0.26	0.30	0.130	0.1235

C_{12}		C_{23}		C_{13}	
Bernoulli	Inv.Fibo.	Bernoulli	Inv. Fibo.	Bernoulli	Inv. Fibo.
0.53	0.31	0.26	0.31	0.130	0.1258
0.53	0.32	0.27	0.317	0.131	0.1280
0.54	0.32	0.27	0.324	0.131	0.1300
0.55	0.33	0.27	0.332	0.131	0.1318
0.55	0.34	0.27	0.339	0.130	0.1335
0.5628	0.35	0.28	0.346	0.130	0.1351
0.57	0.35	0.28	0.352	0.129	0.1365
0.57	0.36	0.28	0.359	0.129	0.1377
0.57	0.36	0.287	0.365	0.128	0.1388
0.58	0.37	0.288	0.370	0.127	0.1398
0.58	0.37	0.290	0.376	0.126	0.1406
0.58	0.38	0.291	0.381	0.125	0.1413
0.58	0.38	0.291	0.386	0.124	0.1418
0.58	0.39	0.292	0.391	0.122	0.1423
0.58	0.39	0.292	0.395	0.121	0.1426
0.58	0.40	0.292	0.399	0.120	0.1427
0.58	0.40	0.292	0.403	0.118	0.1428
0.58	0.41	0.292	0.407	0.117	0.1428
0.58	0.41	0.292	0.410	0.115	0.1426
0.58	0.41	0.29	0.413	0.114	0.1424
0.58	0.42	0.29	0.416	0.112	0.1421
0.58	0.42	0.29	0.418	0.110	0.1416
0.57	0.42	0.29	0.421	0.109	0.1411
0.57	0.42	0.289	0.423	0.107	0.1406

C_{12}		C_{23}		C_{13}	
Bernoulli	Inv.Fibo.	Bernoulli	Inv. Fibo.	Bernoulli	Inv. Fibo.
0.57	0.425	0.287	0.425	0.105	0.1399
0.57	0.427	0.286	0.427	0.104	0.1392
0.57	0.428	0.285	0.428	0.102	0.1384
0.56	0.429	0.284	0.429	0.100	0.1375
0.56	0.430	0.282	0.430	0.099	0.1366
0.56	0.431	0.281	0.432	0.097	0.1356
0.56	0.432	0.280	0.432	0.096	0.1346
0.55	0.433	0.278	0.433	0.094	0.1336
0.55	0.433	0.277	0.433	0.093	0.1325
0.55	0.434	0.275	0.434	0.091	0.1314
0.54	0.434	0.273	0.434	0.089	0.1302
0.54	0.433	0.272	0.434	0.088	0.1290
0.54	0.433	0.270	0.434	0.086	0.1278
0.53	0.433	0.268	0.433	0.085	0.1266
0.53	0.433	0.267	0.433	0.084	0.1253
0.53	0.432	0.265	0.433	0.082	0.1240
0.52	0.432	0.263	0.432	0.080	0.1227
0.520	0.431	0.262	0.431	0.079	0.1215
0.525	0.430	0.260	0.430	0.078	0.1201
0.510	0.429	0.258	0.429	0.077	0.1188
0.515	0.428	0.256	0.428	0.075	0.1175
0.510	0.427	0.255	0.427	0.074	0.1162
0.505	0.426	0.253	0.426	0.073	0.1149
0.500	0.425	0.251	0.425	0.071	0.1135

C_{12}		C_{23}		C_{13}	
Bernoulli	Inv.Fibo.	Bernoulli	Inv. Fibo.	Bernoulli	Inv. Fibo.
0.49	0.423	0.249	0.423	0.070	0.1122
0.49	0.422	0.248	0.422	0.069	0.1109
0.49	0.421	0.246	0.421	0.062	0.1095
0.48	0.419	0.244	0.419	0.067	0.1082
0.48	0.418	0.242	0.418	0.065	0.1069
0.48	0.416	0.241	0.416	0.064	0.1056
0.47	0.41	0.239	0.41	0.063	0.1043
0.47	0.41	0.237	0.413	0.062	0.1030
0.47	0.41	0.23	0.41	0.061	0.1018
0.46	0.41	0.23	0.41	0.060	0.100
0.46	0.41	0.23	0.41	0.059	0.0992
0.46	0.41	0.23	0.41	0.058	0.0980
0.45	0.40	0.23	0.40	0.057	0.0967
0.45	0.40	0.23	0.40	0.056	0.0955
0.45	0.40	0.22	0.40	0.055	0.0943
0.44	0.40	0.22	0.40	0.054	0.0931
0.44	0.40	0.22	0.40	0.054	0.0919
0.44	0.39	0.22	0.39	0.053	0.0908
0.43	0.39	0.22	0.39	0.05	0.0896
0.43	0.39	0.22	0.39	0.05	0.088
0.43	0.39	0.22	0.39	0.05	0.087
0.43	0.38	0.21	0.38	0.05	0.086

Table 8.2: Concurrences between pairs of qubits for Bernoulli and Inverse Fibonacci chains.

Now, we can store the above data in separate text files and plot them as shown in Fig.[8.2] below.

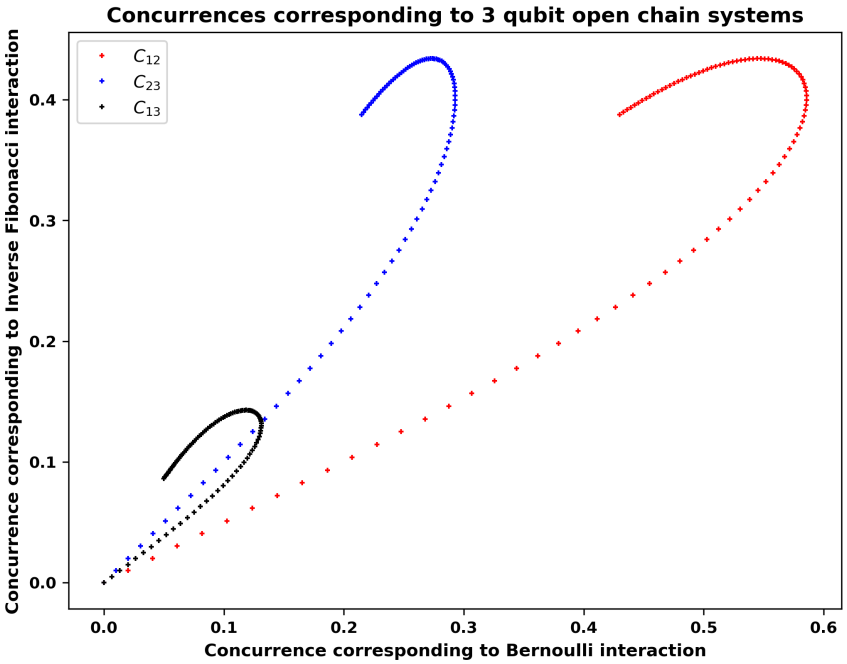


Figure 8.2: Interpolation of concurrences corresponding to inhomogeneous transverse Ising-like 3 qubit open chain system by considering two sets of interaction schemes.

We have tried several curves to fit the data points, and finally found that the floral function could fit it more or less, which can be given as follows in the polar or parametric form,

$$r = \left(3K\cos\left(3\theta+\phi\right)^{p}\right)^{l}. \tag{8.24}$$

We can able to fit the data points with the help of the above curve by just changing the parameters K, ϕ, p & l (which control the number of petals, size of the petals, etc.) as shown below. In Fig.[8.3] we have plotted the function given in Eq.[8.24].

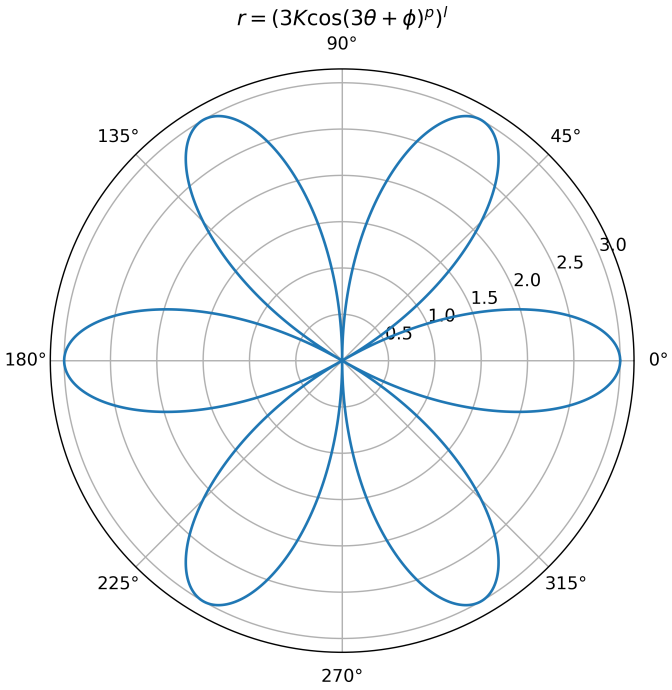


Figure 8.3: Floral Function in Polar Coordinates.

We can change from polar coordinates to cartesian coordinates with the help of the transformation equations given below.

$$x = r\ cos\theta, \tag{8.25}$$

$$y = r\ sin\theta. \tag{8.26}$$

$$\begin{aligned} x^2 + y^2 &= r^2\ cos^2\theta\ + r^2\ sin^2\theta\ , \\ &= r^2(\ cos^2\theta\ + r^2\ sin^2\theta), \\ &= r^2, \\ r &= (x^2 + y^2)^{\frac{1}{2}}. \end{aligned} \tag{8.27}$$

$$\begin{aligned} \frac{y}{x} &= \frac{r\ sin\theta}{r\ cos\theta}, \\ &= tan\theta, \\ \theta &= tan^{-1}\left(\frac{y}{x}\right). \end{aligned} \tag{8.28}$$

So that Eq.[8.24] boils down to,

$$\left(x^2 + y^2\right)^{\frac{1}{2}} - \left(3K\cos\left(3\tan^{-1}\left(\frac{y}{x}\right) + \phi\right)^p\right)^l = 0 \tag{8.29}$$

Now we can plot the floral function Eq.[8.29] given in cartesian coordinates (by choosing the fitting parameters $K = 1,\ \phi = 0,\ p = 1,$ and $l = 1$), which is shown in Fig.[8.4] below.

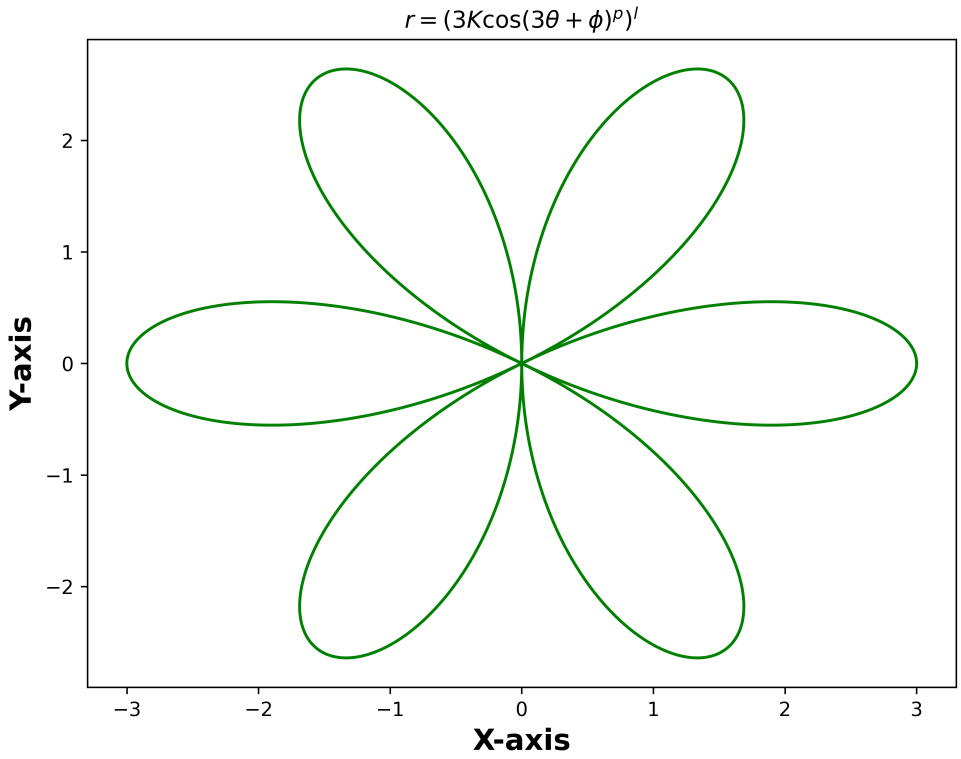


Figure 8.4: Floral Function in Cartesian Coordinates.

We can see the effects of these fitting parameters as follows.

Effect of fitting parameter K

We can observe from the Fig.[8.5] that the fitting parameter K can help to increase or decrease the size of the floral petals slightly.

$$K = 2,$$
$$\phi = 0,$$
$$p = 1,$$
$$l = 3.$$

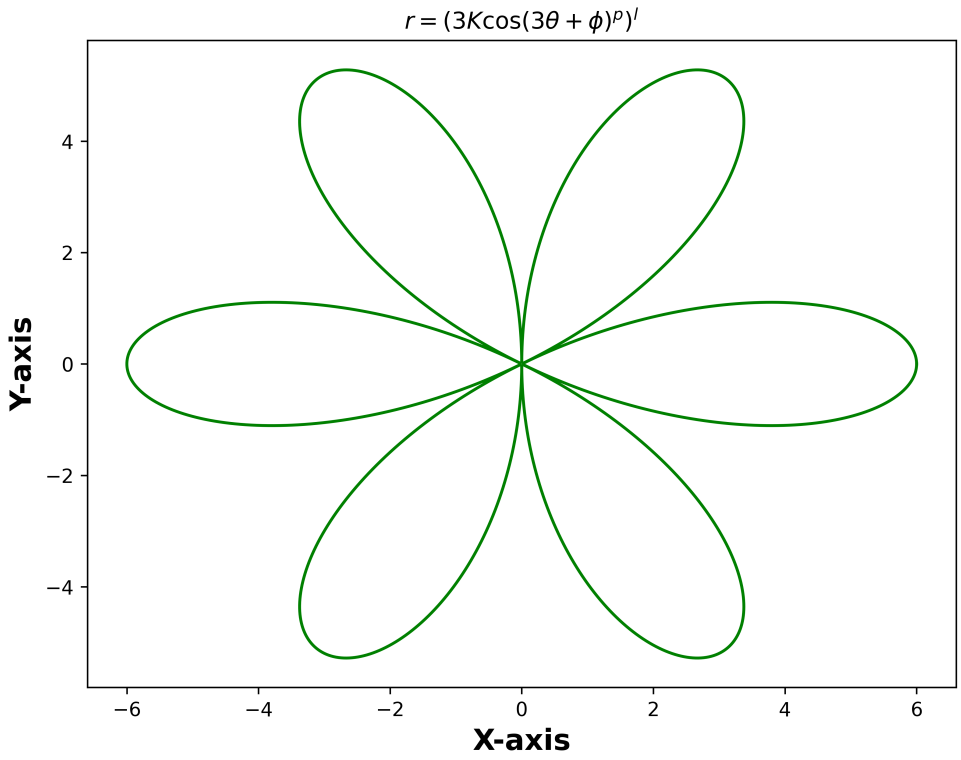


Figure 8.5: Effect of fitting parameter K in floral function.

Effect of fitting parameter l

We can observe from the Fig.[8.6] that the fitting parameter l can help to increase or decrease the size of the floral petals by a larger value compared to K.

$$K = 1,$$

$$\phi = 0,$$

$$p = 1,$$

$$l = 3.$$

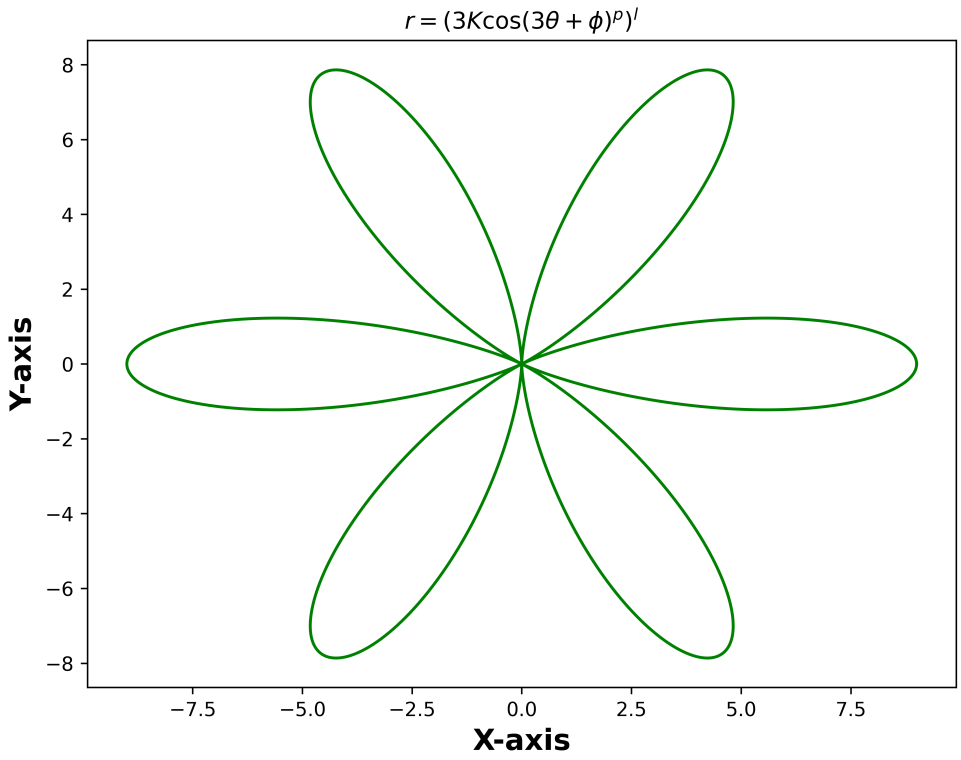


Figure 8.6: Effect of fitting parameter l in floral function.

Effect of fitting parameter ϕ

We can observe from the Fig.[8.7] that the fitting parameter ϕ can help to orient the floral petals and flatten it.

$$K = 1,$$

$$\phi = 1,$$

$$p = 1,$$

$$l = 3.$$

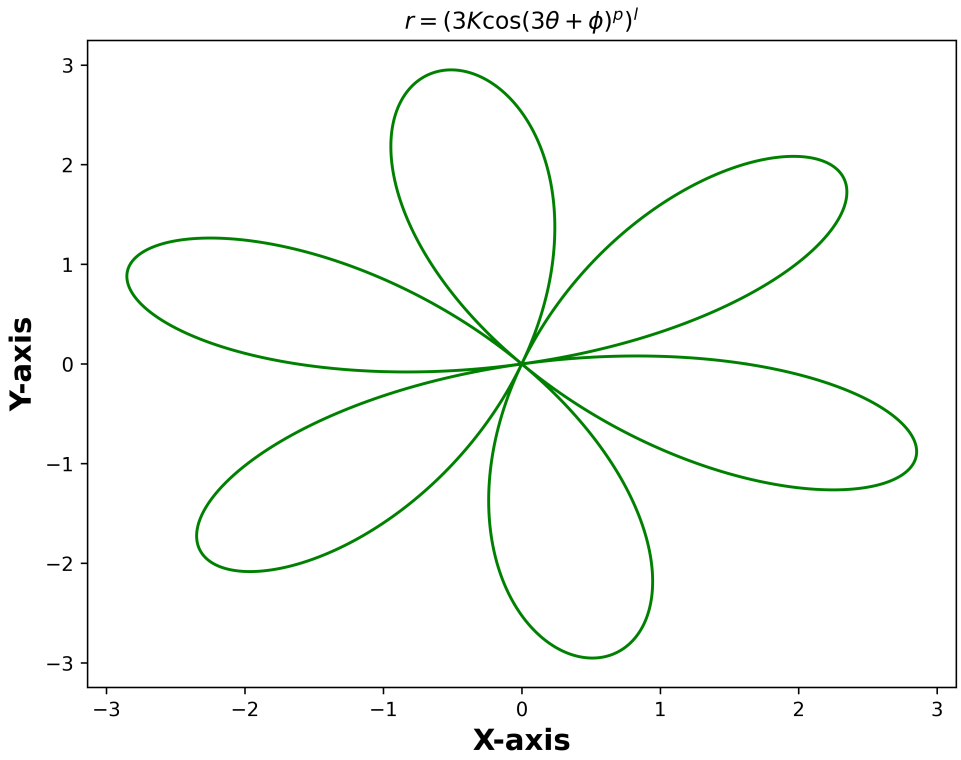


Figure 8.7: Effect of fitting parameter ϕ in floral function.

Now, we can adjust the fitting parameters for our requirements to fit the data points corresponding to the concurrences as follows.

8.3.2.1 Fitting of C_{12} data points with the help of floral function.

Fitting parameters

For fitting of C_{12} data points the constant parameters are chosen as,

$$K = -0.3,$$

$$\phi = 1.23,$$

$$p = 1.4,$$

$$l = 3.2.$$

By choosing the fitting parameters mentioned above, we can able to fit the C_{12} data points along one of the petals of this floral structure.

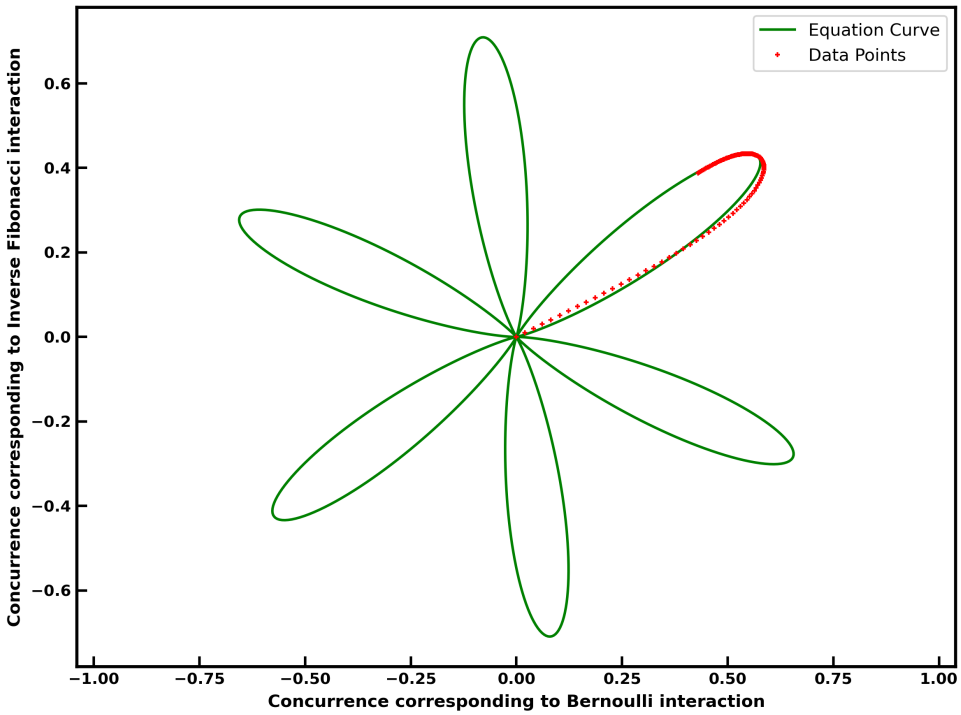


Figure 8.8: Fitting the C_{12} data with the help of floral function.

We can zoom in on the fitted part of Figure 8.8 and see Figure 8.9.

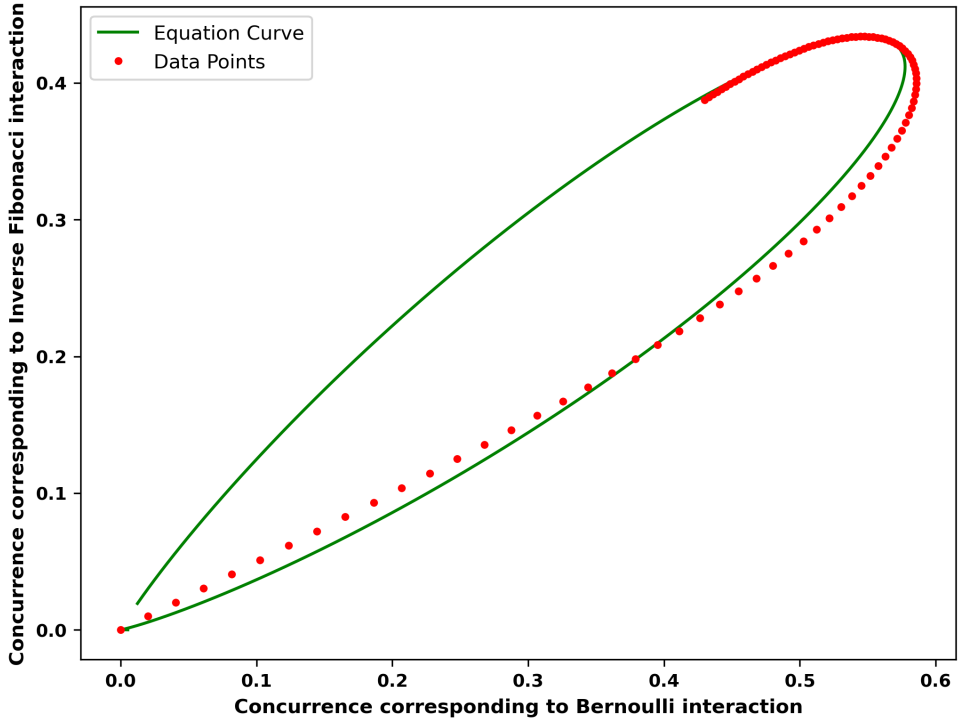


Figure 8.9: Zoomed in view of fitted portion of the C_{12} data.

8.3.2.2 Fitting of C_{23} data points with the help of floral function.

Fitting parameters

For fitting of C_{23} data points the constant parameters are chosen as,

$$K = 0.192,$$

$$\phi = 0.19,$$

$$p = 3,$$

$$l = 1.20.$$

By choosing the fitting parameters mentioned above, we can now fit the C_{23} data points along one of the petals of the modified floral structure.

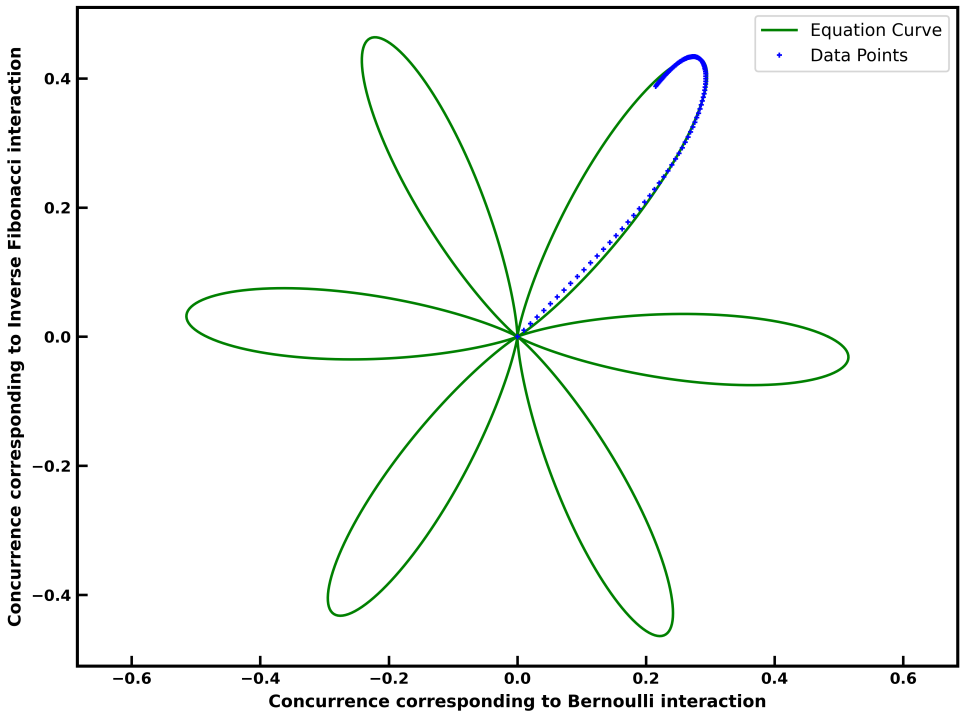


Figure 8.10: Fitting the C_{23} data with the help of floral function.

We can zoom in on the fitted part of Fig.[8.10] and see Fig.[8.11].

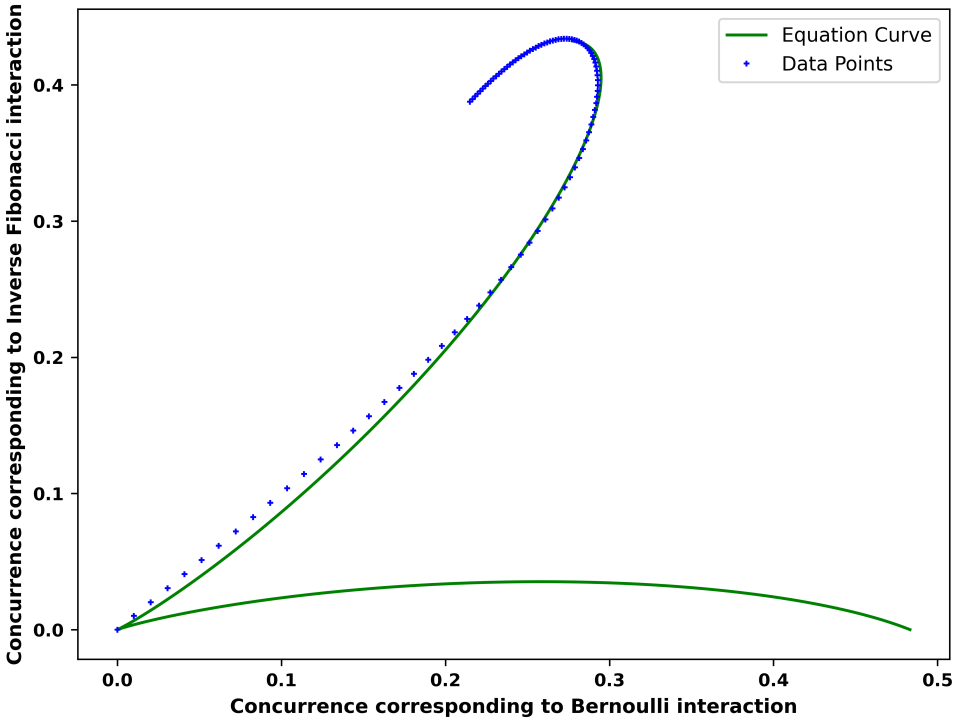


Figure 8.11: Zoomed in view of fitted portion of the C_{23} data.

8.3.2.3 Fitting of C_{13} data points with the help of floral function.

Fitting parameters

For fitting of C_{13} data points the constant parameters are chosen as,

$$K = -0.2,$$

$$\phi = 0.74,$$

$$p = 1.4,$$

$$l = 3.3.$$

By choosing the fitting parameters mentioned above, we can now fit the C_{13} data points along one of the petals of the modified floral structure.

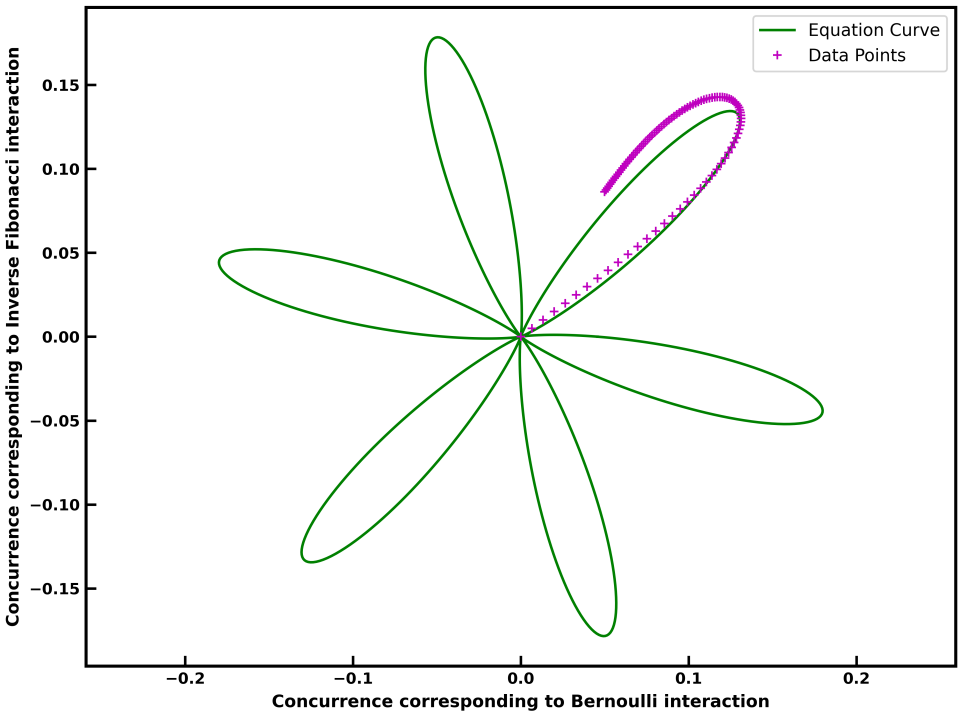


Figure 8.12: Fitting the C_{13} data with the help of floral function.

We can zoom in on the fitted part of Fig.[8.12] and see Fig.[8.13].

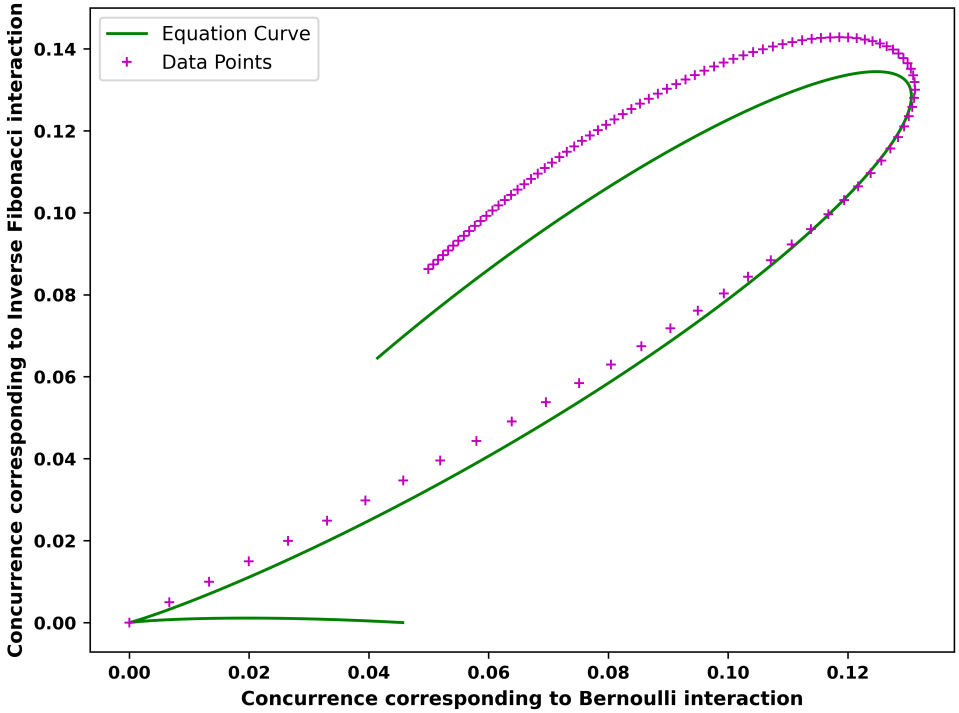


Figure 8.13: Zoomed in view of fitted portion of the C_{13} data.

From these observations, we can say that the concurrences corresponding to the nearest neighbour qubit pairs can be more accurately fitted by the floral function as provided above in Eq.[8.29]. Therefore, for the three-qubit case, the concurrences corresponding to the inverse Fibonacci interaction($C_{inv.Fibo}$) and that for the Bernoulli type interaction ($C_{Bernoulli}$) of the Ising-like model are related as given below with the adjusted parameters.

$$\left((C_{Bernoulli})^2 + (C_{inv.Fibo})^2\right)^{\frac{1}{2}} - \left(3K\cos\left(3\tan^{-1}\left(\frac{C_{inv.Fibo}}{C_{Bernoulli}}\right) + \phi\right)^p\right)^l = 0. \quad (8.30)$$

8.4 Chapter Summary

In this chapter, we have started from the relation that exists between the Fibonacci and Bernoulli numbers. We were aiming to study how this relation affects the entanglement between the qubits corresponding to the two random transverse Ising-like systems involving the Bernoulli and inverse Fibonacci numbers as their interaction strengths. We had the data corresponding to the ground state energies and concurrences for the three-qubit open chain systems. We have analysed those results in this chapter. Now, the analytical solution is not possible because of the randomness in the system due to the random selection of the interaction energies from the two number series, and also, the interaction energies vary from site to site. However, we have succeeded in solving our problem numerically, and the technique of interpolation helped us to find relations between the results obtained from the two models.

★

Bibliography

[1] M. Plischke and B. Bergersen, *Equilibrium Statistical Physics*, 3rd ed. (World Scientific, Singapore, 2006).

[2] S. G. Brush, History of the Lenz-Ising model, *Reviews of Modern Physics*, **vol. 39** (no. 4), page no. 883 (1967).

[3] P. Pfeuty, The one-dimensional Ising model with a transverse field, *Annals of Physics*, **vol. 57** (no. 1), page no. 79 (1970).

[4] N. Zettili, *Quantum Mechanics: Concepts and Applications*, 2nd ed. (John Wiley & Sons, Chichester, 2009).

[5] W. Heisenberg, Zur Theorie des Ferromagnetismus, *Zeitschrift für Physik*, **vol. 49** page no. 619 (1928).

[6] M. Kardar, *Statistical Physics of Particles* (Cambridge University Press, Cambridge, 2014).

[7] S. Blundell, *Magnetism in Condensed Matter* (Oxford University Press, Oxford, 2001).

[8] J. B. Parkinson and D. J. Farnell, *An Introduction to Quantum Spin Systems* (Springer, Berlin, 2010).

[9] H. Bethe, Zur Theorie der Metalle, *Zeitschrift für Physik*, **vol. 71**, page no. 205 (1931).

[10] S. Katsura, Statistical mechanics of the anisotropic linear Heisenberg model, *Physical Review* **vol. 127** (no. 5), page no. 1508 (1962).

[11] L.E. Sigler (Transl.), *Fibonacci's Liber Abaci: Leonardo Pisano's Book of Calculation*, (Springer-Verlag, New York, 2002).

[12] I. Gosai, Fibonacci Sequence and It's Applications, *International Journal of Research and Analytical Reviews (IJRAR)*, **vol. 6** (no. 2) page no. 2 (June 2019).

[13] H. Tsunetsugu and K. Ueda, Ising spin system on the Fibonacci chain, *Physical Review B*, **vol. 36** (no. 10) page no. 5639 (1987).

[14] K. Hida, Renormalization Group Studies of the Fibonacci Quantum Spin Chains, *Journal of the Physical Society of Japan*, **vol. 73** (no. 6), page no. 1534 (2004).

[15] F. Samavat, M. H. Tavakoli, S. Habibi, B. Jaleh, and P. T. Ahmad, Quasicrystals, *Open Journal of Physical Chemistry*, **vol. 2** (no. 1), page no. 7 (2012).

[16] "Penrose Tiling" image, available at: https://images.app.goo.gl/5kFPdT4SoEM8fTHf9,

[17] R. Horodecki, P. Horodecki, M. Horodecki, and K. Horodecki, Quantum entanglement, *Rev. Mod. Phys.* **81**, page no.865 (2009).

[18] A. Einstein, B. Podolsky, and N. Rosen, Can quantum-mechanical description of physical reality be considered complete? *Physical Review*, **vol. 47**, (no. 10), page no. 777 (1935).

[19] W. K. Wootters, Entanglement of formation of an arbitrary state of two qubits, *Physical Review Letters*, **vol. 80** (no. 10), page no. 2245 (1998).

[20] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, Cambridge, 2010).

[21] A. Wehrl, General properties of entropy, *Reviews of Modern Physics*, **vol. 50** (no. 2), page no. 221 (1978).

[22] X.-L. Zong, H.-H. Yin, W. Song, and Z.-L. Cao, Monogamy of Quantum Entanglement, *Frontiers in Physics*, **10** (2022).

[23] K. M. O'Connor and W. K. Wootters, Entangled rings, *Physical Review A*, **vol. 63** (no. 5) (2001).

[24] R. P. Feynman, "Forces in molecules," *Physical Review*, **vol. 56** no. 4, page no. 340 (1939).

[25] J. Bernoulli, *Ars Conjectandi, Opus Posthumum*, Thurnisiorum Fratrum, Basel, 1713.

[26] M. Abramowitz and I. A. Stegun (Eds.), *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, (National Bureau of Standards, Applied Mathematics Series 55, U.S. Government Printing Office, Washington, D.C., 1964).

[27] G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers*, 5th ed. (Pergamon Press, Oxford, 1960).

[28] P. Štelmachovič and V. Bužek, Quantum-information approach to the Ising model: Entanglement in chains of qubits, *Physical Review A*, **vol. 70** page no. 032313 (2004).

[29] M. A. Nielsen, The Fermionic canonical commutation relations and the Jordan-Wigner transform, *arXiv preprint arXiv:quant-ph/0504097* (2005).

[30] S. R. K. Iyengar and R. K. Jain, *Numerical Methods* 3rd ed., (New Age International (P) Limited, Publishers, New Delhi, 2015).

[31] M. S. Ramkarthik and P. D. Solanki, *Numerical Recipes in Quantum Information Theory and Quantum Computing: An Adventure in FORTRAN 90* (CRC Press, Boca Raton, 2022).